An Experimental Study on Federated Equi-Joins

Shuyuan Li, Yuxiang Zeng, Yuxiang Wang, Yiman Zhong, Zimu Zhou, *Member, IEEE*, Yongxin Tong, *Member, IEEE*

***Abstract*—Data federation has emerged as a novel database system enabling collaborative queries across mutually distrusted data owners. Federated equi-join, a commonly used operation in data federation, combines relations from distinct data owners while preserving their data privacy. Due to the wide applications of this query, many solutions to federated equi-joins have been proposed. However, it is still challenging for practitioners to choose the most appropriate algorithm due to various reasons, including incomplete evaluation protocols (*e.g.*, lack of evaluating multi-way equi-joins), under-explored performance metric (main memory usage), and absence of a standardized comparison. Motivated by this reason, this paper conducts a comprehensive experimental study and builds a new benchmark, called FEJ-Bench, for federated equi-joins. The experimental study and the benchmark consist of eight state-of-the-art algorithms and five datasets. Our evaluation reveals the query efficiency ranking, its impact factors, and potential research opportunities. Finally, we open-source FEJ-Bench on GitHub, which is the first benchmark for federated equi-joins. Our findings aim to guide researchers and practitioners in deploying federated equi-joins in practice.**

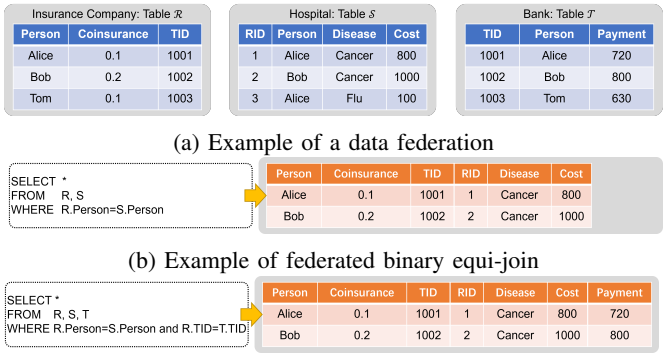
***Index Terms*—Equi-join, data federation, secure multi-party computation, benchmark.**

I. Introduction

In today's data-driven world, the demand for efficient data collaboration services across multiple data owners has become increasingly crucial. However, in the real world, these data owners often experience a state of mutual distrust, stemming from factors such as safeguarding data confidentiality and adhering to legal and regulatory compliance requirements (*e.g.*, GDPR). Consequently, it becomes imperative for the service providers to prioritize the security of these data owners while facilitating their collaborative queries and analytics.

To enable such services, a novel type of database system called "*data federation*" [1]–[3] has emerged as a potential solution in recent years. A data federation can be perceived as a union of datasets held and autonomously managed by mutually distrustful data owners. When querying the data federation, often referred to as a *federated query* [3], it necessitates secure and efficient processing that is jointly conducted by the data owners over their respective datasets.

The *federated equi-join* is commonly used in a data federation [1], [3]. It enables the combination of two or more relations held by distinct data owners, depending on the equality of attribute values specified in the join condition. Moreover, the query user remains unaware of any information beyond the query result itself or any details that can be directly inferred from it. Similarly, any data owner is limited to the knowledge of his own input and output, and has no clue about the others' sensitive data. Federated equi-joins have been applied in several real-world applications [4], such as passenger watch-list verification and cross-platform investigation. A toy example of such a query is as follows.



(a) Example of a data federation

(b) Example of federated binary equi-join

(c) Example of federated multi-way equi-join

Fig. 1: Federated equi-join query over a data federation.

Example 1: Fig. 1 illustrates a simplified application scenario of federated equi-joins. In this scenario, the tax bureau plans to investigate an insurance company and particularly checks the payments of its insured patients. This investigation involves three tables $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{T}$ from the insurance company, hospital, and bank, respectively. The first rows of the tables in Fig. 1a represent the attributes of these relations. Two instances of the federated equi-joins, which involve different numbers of tables, are depicted in Fig. 1b and 1c, respectively.

Many existing studies have designed novel solutions to answer a federated equi-join, and their proposed solutions can be classified into four categories. The first three are inspired by the seminal ideas of equi-joins over plaintext data: *nested-loop* based federated equi-joins [1], [5], [6], *sort-merge* based federated equi-joins [4], [7], and *hash* based federated equi-joins [8]. The other line of research [9], [10] leverages the rich techniques of *private set intersection (PSI)* [11] that securely computes the intersection of two attribute sets (*e.g.*, primary keys) from different data owners.

Motivation. Despite the research attention, it is still hard for practitioners to select a proper solution due to these reasons.

- **Incomplete evaluation protocols.** The previous evaluations mainly focused on the impact of the input size while ignoring other crucial factors, such as data distribution (*e.g.*, join selectivity) and query types (*e.g.*, equi-joins over primary keys PK or foreign keys FK). Moreover, most of these studies [1], [5], [7], [8], [10] solely evaluated the equi-joins over two relations (*i.e.*,

S. Li, Y. Zeng, Y. Wang, Y. Zhong, and Y. Tong are with the State Key Laboratory of Software Development Environment and Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, School of Computer Science, Beihang University, China. E-mail: {lishuyuan, yxzeng, yuxiangwang, yeeman, yxtong}@buaa.edu.cn.

Z. Zhou is with the School of Data Science, City University of Hong Kong, Hong Kong SAR, China. E-mail: zimuzhou@cityu.edu.cn.

federated binary equi-joins), overlooking the assessment of federated multi-way equi-joins. However, these factors are common in real-world applications and have been extensively tested in joins over plaintext data [12], [13].

- **Under-explored performance metric.** Most previous empirical studies primarily evaluate the query processing time and communication cost, overlooking the main memory usage. Only [6] reported an instance, where their evaluated method ran out of main memory with just 30k tuples in a relation. This issue remained unnoticed in other evaluations, since the tested queries may involve additional selection or aggregation operations over equi-joins, resulting in a limited intermediate table size for joins that were not as large as the input size. Notice that, secure computations usually require high memory usage. According to the example in [14], a long integer, which normally takes 8 bytes of main memory in plaintext, requires 8192 bytes by using the prevalent secure protocol, garbled circuit [11]. Thus, the memory usage is also crucial in assessing the overall query performance.
- **Absence of a standardized comparison.** Previous methods lack a unified comparison within the same experimental benchmark. One reason is that their implementations are either unavailable or non-uniform (see Table I). Moreover, their methods are evaluated on nearly completely different datasets. For example, the TPC-H [15] is the only public dataset shared between these studies [4], [9].

Contribution. To overcome these limitations, we conduct comprehensive evaluations by comparing 8 federated equi-join algorithms on 4 real datasets and 1 benchmark dataset TPC-H, and make the following observations and contributions:

(1) In general, the efficiency ranking we have observed is "PSI $>$ hash $>$ sort-merge $>$ nested-loop" (from best to worst). Here, the *PSI* based methods only support primary-key-to-primary-key federated equi-joins, where the join condition assumes unique attribute values. For most of the other cases, the *hash* based method [8] is the optimal choice.

(2) Join selectivity, which has been overlooked by previous evaluation protocols, can overthrow this efficiency ranking. Specifically, when the federated equi-join involves foreign keys and takes a high selectivity, the *nested-loop* based [6] and *sort-merge* based [7] methods can outperform the *hash* based method [8] for federated binary equi-joins and federated multi-way equi-joins respectively, and hence change the optimal choice.

(3) We have demonstrated the join order optimization can potentially improve the efficiency of federated multi-way equi-joins by a large margin. In our experiment, the query processing time of *hash* based and *sort-merge* based methods can be notably reduced by 27% to 62% when replacing with the optimal join order. Thus, we identify the join order optimization as a promising future research opportunity, although previous work [9] feels opposite.

(4) The memory usage, which is under-explored in existing work, can become the efficiency bottleneck for most algorithms. For example, when processing federated equi-joins with foreign keys over the real dataset Amazon [16] that has 120k and 30k tuples for two relations, the main memory usage of all the *nested-loop* based algorithms exceed 1TB, while the other methods take over 150GB. In contrast, the plaintext nested-loop based equi-join only takes 2GB.

(5) We have built an open-sourced Benchmark for Federated Equi-Joins, called FEJ-Bench. To foster future research studies, we have made both code and datasets public on GitHub [17]. This benchmark is the *first* of its kind for federated equi-joins, to the best of our knowledge.

Road Map. In the rest of this paper, we first present the definition of the federated equi-join and related secure primitives in Sec. II. Next, we introduce existing federated equi-join algorithms in Sec. III. Then, we present our experimental setup in Sec. IV and evaluation result in Sec. V. Finally, we identify future direction in Sec. VI and conclude in Sec. VII.

II. Preliminary

This section introduces the query and secure primitives.

A. Problem Statement

Definition 1 (Data Owner): A data downer (*a.k.a.* data silo [18]), denoted by o, holds a relational table T_o with n tuples as their own dataset.

Based on the definition of a data owner, a data federation can be defined as follows.

Definition 2 (Data Federation [1], [19]): The data federation F is a collection of m data silos $F = \{o_1, o_2, \cdots, o_m\}$, each of whom holds and manages a relational table $\mathcal{T}_{o_i}$.

In Def. 2, the federation has m different relational tables $\mathcal{T}_{o_1}, \cdots, \mathcal{T}_{o_m}$ and also knows the schema of these tables in advance. Equi-joins over two relations under a data federation are abbreviated as *federated binary equi-join* by us and also known as oblivious equi-joins in existing work [4], [7]. According to the number of relations involved in the query, federated equi-joins include *federated binary equi-joins* and *federated multi-way equi-joins*, which are defined as follows.

Definition 3 (Federated Binary Equi-Join): Given a data federation F of two tables $\mathcal{R}$ and $\mathcal{S}$ held by the data owners o_i and o_j (*i.e.*, $\mathcal{R} = \mathcal{T}_{o_i}$ and $\mathcal{S} = \mathcal{T}_{o_j}$), and a join condition $\mathcal{R}.A = \mathcal{S}.B$ over specific attributes A on table $\mathcal{R}$ and attributes B on table $\mathcal{S}$, the federated binary equi-join $\mathcal{R} \bowtie \mathcal{S}$ aims to retrieve all pairs of tuples $r \in \mathcal{R}$ and $s \in \mathcal{S}$ that have equal values in their attributes $r.A$ and $s.B$:

$$\mathcal{R} \bowtie \mathcal{S} = \{(r, s) \mid r \in \mathcal{R}, s \in \mathcal{S}, r.A = s.B\}$$

and meet the following security requirement:

- **Attacker Model**. Each data owner is a semi-honest (*a.k.a.* honest-but-curious) adversary attacker [11]. That is, each data owner will honestly execute the pre-specific query processing protocol but may attempt to infer the other's sensitive data during the query processing.
- **Security Constraint**. The query user can only know the join result (or any information can be directly deduced from the join result). Each data owner can only know their own input/output during the query processing.

Definition 4 (Federated Multi-Way Equi-Join): Given a data federation F of m tables $\mathcal{T}_1, \cdots, \mathcal{T}_m$ held by the data owners $o_1, \cdots, o_m$, and a collection C of equi-join conditions

$\{\mathcal{T}_i.A_i = \mathcal{T}_j.B_j\}$ over specific attributes A_i on table $\mathcal{T}_i$ and attributes B_j on table $\mathcal{T}_j$, the federated multi-way equi-join $\mathcal{T}_1 \bowtie \cdots \bowtie \mathcal{T}_m$ aims to retrieve all combinations of tuples $(t_1, \cdots, t_m)$ that satisfy all the equi-join conditions in C:

$$\mathcal{T}_1 \bowtie \cdots \bowtie \mathcal{T}_m = \{(t_1, \cdots, t_m) \mid \forall (i,j) \in C, t_i.A_i = t_j.B_j\}$$

and meet the same security requirement in Def. 3.

Based on the above definitions, a federated multi-way equi-join can be decomposed into a series of federated binary equi-joins without leaking the intermediate results of those binary equi-joins to either the query user or the data owners.

Remark. The attackers in federated equi-join queries are semi-honest, which is a commonly used assumption in existing work [1], [4]–[9]. The other type of attacker model is the malicious model [11] that assumes the corrupted data owners of the attacker can deviate from the pre-specified query processing procedure (*e.g.*, by changing their inputs). Although the malicious attacker model is often more challenging than the semi-honest attacker model, only a few work [20], [21] has studied the federated equi-joins under this setting. Thus, we mainly consider the semi-honest attacker in this work.

B. Secure Multi-Party Computation Primitive

To compute the query result, secure multi-party computation (SMC) techniques have been widely applied in existing work to meet the security requirement. To better understand existing work, we introduce related SMC primitives in the following.

Garbled Circuit. Garbled Circuit (GC) is probably the most famous technique of SMC [11]. A GC protocol usually aims to securely evaluate a function $F(x, y)$ based on the private inputs x and y from two mutually distrusted parties. In the seminal Yao's millionaire problem [11], the function $F(x, y)$ is to compare whether x is no smaller than y without leaking the values of x and y, where x and y denote the actual wealth of the millionaires, Alice and Bob. To solve this problem, Yao's GC protocol [11] has been widely studied. A typical usage of Yao's GC protocols in federated equi-joins is to check the join condition by the equality of the involved attribute values. Notice that, Yao's GC protocol is highly generic, since it supports all the discrete functions F that can be converted into a fixed-size circuit [11].

Private Set Intersection. By contrast, private set intersection (PSI) is one of the specific functionalities that require a tailored protocol. Intuitively, PSI usually computes the intersection of two sets X and Y with unique elements from Alice and Bob, respectively. In the end, Alice receives the computation result $X \cap Y$, and Bob knows nothing except for his input Y. Moreover, recent work on PSI focuses on the problem of *circuit-PSI*. In a circuit-PSI, the result of $X \cap Y$ is secret-shared between Alice and Bob, while neither Alice nor Bob knows the actual output. The circuit-PSI protocols (*e.g.*, OPPRF-PSI [22]) have been used in secure join processing [9], [20] to improve the query efficiency (see Sec. III-E for more details).

SMC Implementation Tools. There are two ways to implement and apply previous SMC protocols in a federated equi-join algorithm, *i.e.*, using a *SMC library* or *SMC compiler*.

- **SMC Library.** A SMC library encapsulates many basic protocols, such as Yao's garbled circuits, and provides a user-friendly API interface. A typical SMC library, which has been commonly used in secure query processing, is ABY [23]. The SMC library of ABY is open-sourced and mainly supports secure computations across two parties.
- **SMC Compiler.** A SMC compiler first defines its high-level programming language, then uses this language to code secure computations, and finally behaves like a compiler to generate the executable program. ObliVM [24] and Obliv-C [25] are two popular SMC compilers based on the Java/C-like programming language, and have been used in federated equi-join algorithms [1], [6].

By contrast, the benefit of a SMC compiler is enabling users to code without caring about protocol details, while a SMC library is more likely to achieve the expected efficiency. To ensure a uniform and fair comparison, we use the SMC library ABY [23] to implement the following methods.

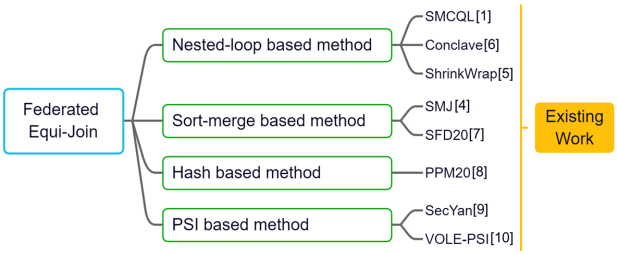


Fig. 2: Taxonomy of federated equi-join algorithms.

III. Federated Equi-Join Algorithm Overview

In this section, we first introduce our taxonomy and a general framework of existing solutions. Next, based on this taxonomy, we review existing federated binary equi-join solutions from four categories, *i.e.*, nested-loop based, sort-merge based, hash based, and PSI based methods. Then, we present extensions to federated multi-way equi-joins. Finally, we provide a comparison of these solutions.

A. Taxonomy and General Framework

Taxonomy. Equi-joins (over plaintext data) have been extensively studied in the field of databases [12]. The text books commonly classifies equi-join algorithms into three kinds: nested-loop join, sort-merge join, and hash join [26]. In fact, many federated equi-join algorithms are designed based on these principles. Additionally, recent developments on Private Set Intersection (PSI) protocols have inspired another algorithm category to process federated equi-joins at scale. Therefore, as shown in Fig. 2, we classify existing federated equi-join algorithms into four categories: *nested-loop* based, *sort-merge* based, *hash* based, and *PSI* based methods. We also indicate the specific algorithms, such as SMCQL [1] and Conclave [6], that fall under each category.

General Framework. Although existing methods are inspired by different ideas, their processing workflows share several common steps and hence can be represented by the general framework depicted in Fig. 3. Specifically, a query user submits their query request to the service provider (*a.k.a.* data broker in [1]). The service provider then acts as a query coordinator to assist the data owners in the query processing, which generally consists of the following steps:

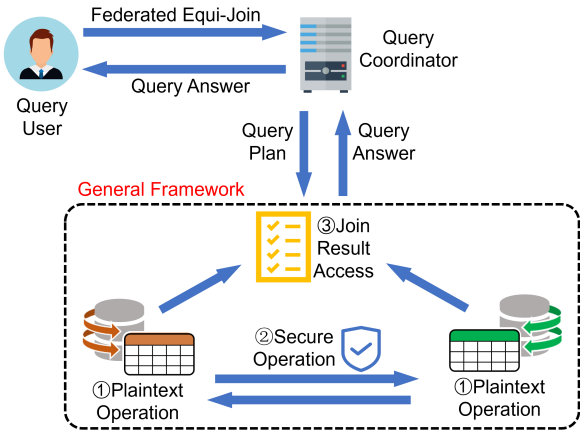


Fig. 3: General framework of federated equi-joins.

(1) Each data owner locally performs plaintext operations over their dataset, so no privacy leakage occurs at this step.
(2) The data owners collaboratively perform secure operations over their datasets, and data security needs to be carefully preserved at this step.
(3) The data owners generate the join result and send it back to the query user through the service provider.

Remark. In practice, there are cases where the result of joining two relations is regarded as an intermediate result and cannot be directly revealed to the query user due to the security requirement. Instead, the intermediate join result is kept in a secret-shared form among the data owners [9].

Based on these categories, we will now proceed to review existing solutions in the following subsections.

B. Nested-Loop based Federated Binary Equi-Joins

Main Idea. There are no plaintext operations at the first step. The main idea of the second step is to securely compare all pairwise tuples (r, s) based on their attributes in the join condition using a nested-loop. The last step appends a tuple pair (either (r, s) or a dummy tuple) to the join answer based on the comparison result.

Basic Algorithm. Alg. 1 illustrates an example of this kind, namely SMCQL [1]. In line 4, SMCQL implements a garbled circuit based protocol for the secure equality comparison. If the compared attribute values are equal, SMCQL concatenates the attribute values of tuples r and s, eliminate the repeated attributes, and append this tuple to the result Res that is stored in the oblivious RAM (ORAM) [11] in lines 5-6. ORAM is a cryptographic primitive that simulates the Random Access Memory (RAM) model and conceals the access pattern of sensitive data, such as the join result Res. Finally, dummy tuples are padded into the join result Res in line 7. The padding of dummy tuples fulfills the security requirement (*a.k.a.* obliviousness [27] in this context).

Variant Method. Conclave [6] is also designed based on the nested-loop in lines 2-3. However, there are two major differences between Conclave and SMCQL: (1) in Conclave, a dummy tuple will be appended to the query result if the attributes in line 4 are unequal, and (2) the join result is stored using a circuit model rather than the ORAM model.

Algorithm 1: Nested-loop based method SMCQL [1]

Input: Two data owners with relations $\mathcal{R}$ and $\mathcal{S}$
Output: The join result $Res = \mathcal{R} \bowtie_{\mathcal{R}.A=\mathcal{S}.B} \mathcal{S}$

1 $k \leftarrow 0$;
2 **foreach** *tuple* r *in* $\mathcal{R}$ **do**
3 **foreach** *tuple* s *in* $\mathcal{S}$ **do**
4 **if** *attributes* $r.A$ *(securely) equal to* $s.B$ **then**
5 ORAM $Res[k] \leftarrow$ CONCAT(r, s);
6 $k \leftarrow k + 1$;
7 append $(|\mathcal{R}| \cdot |\mathcal{S}| - k)$ dummy tuples to ORAM Res;

Remark. The time complexity of a nested-loop based federated binary equi-join algorithm is typically $O(n^2(T_{cmp} + T_{res}))$, where T_{cmp} and T_{res} denote the time cost of lines 4 and 5 respectively, and n is the number of tuples in each relation. In the ORAM model, T_{res} is $O(\log^3 n)$ [1], whereas in the circuit model, it is often assumed to be $O(1)$ [6].

C. Sort-Merge based Federated Binary Equi-Joins

Compared with nested-loop based methods, sort-merge based methods can reduce the number of secure comparisons.

Main Idea. In the first step of our general framework, both relations $\mathcal{R}$ and $\mathcal{S}$ are sorted based on the attributes in the join condition. Next, by merging the two sorted tuples using a similar procedure in merge sort, we reduce the number of secure comparisons at the second step of our framework.

Routine Method. Alg. 2 presents a federated binary equi-join algorithm SMJ based on the routine of sort-merge join [26]. The algorithm begins by locally sorting the relations $\mathcal{R}$ and $\mathcal{S}$ by the data owners. Then, in line 4, it securely compares the attributes values of $\mathcal{R}[i]$ and $\mathcal{S}[j]$. If they are equal in line 5, SMJ enumerates all tuples in $\mathcal{S}$ that share the same attribute value in the join condition (lines 6-9). In case of inequality, a dummy tuple is appended to the join result Res to fulfill the security requirement. The algorithm proceeds by moving to the next tuple in either $\mathcal{R}$ or $\mathcal{S}$ based on the result eq of the secure comparison (lines 12-15). When retrieving a tuple from one relation, SMJ also pulls a dummy tuple from the other relation to prevent any potential inference of sensitive information based on the tuple access in $\mathcal{R}$ and $\mathcal{S}$.

Variant Method. Krastnikov *et al.* [7] also propose a sorting based federated equi-join algorithm (named as SFD20 in our paper, as it lacks a designated name). The *main idea* of SFD20 is to create two expanded relations $\mathcal{R}'$ and $\mathcal{S}'$ from the original relations $\mathcal{R}$ and $\mathcal{S}$, respectively. This construction enables a simple row-to-row join (*i.e.*, concatenation) to produce the join result. The *key step* is securely computing the expanding numbers α_t and α_s for each tuple $t \in \mathcal{R}$ and $s \in \mathcal{S}$. Here, α_t denotes the number of tuples in $\mathcal{S}$ that share the same attribute values in the join condition with the tuple t, while α_s is defined in a similar way. To compute these numbers, SFD20 employs a sorting network [28] to securely sort all the tuples in $\mathcal{R}$ and $\mathcal{S}$ based on their attribute values in the join condition. Finally, when aligning the expanded tuples and concatenating them to generate the join result, an additional sort is applied to the union of expanded tuples.

Algorithm 2: Sort-merge based method SMJ [4]

Input: Two data owners with relations $\mathcal{R}$ and $\mathcal{S}$
Output: The join result $Res = \mathcal{R} \bowtie_{\mathcal{R}.A=\mathcal{S}.B} \mathcal{S}$

1 locally sort $\mathcal{R}$ and $\mathcal{S}$ based on $\mathcal{R}.A$ and $\mathcal{S}.B$;
2 $i \leftarrow 0, j \leftarrow 0$;
3 **while** $i < n$ *and* $j < m$ **do**
4 $eq \leftarrow$ compare $\mathcal{R}[i].A$ and $\mathcal{S}[j].B$, $k \leftarrow j$;
5 **if** eq *is 0 (i.e., equal)* **then**
6 **while** eq *is 0* **do**
7 append CONCAT$(\mathcal{R}[i], \mathcal{S}[k])$ to Res;
8 $k \leftarrow k + 1$, pull a dummy tuple from $\mathcal{R}$;
9 $eq \leftarrow$ compare $\mathcal{R}[i].A$ and $\mathcal{S}[k].B$;
10 **else**
11 append a dummy tuple to Res;
12 **if** $eq < 0$ **then**
13 $i \leftarrow i + 1$, pull a dummy tuple from $\mathcal{S}$;
14 **else**
15 $j \leftarrow j + 1$, pull a dummy tuple from $\mathcal{R}$;

Remark. The complexity of this algorithm category is closely tied to the complexity of the underlying sort. For example, since the bitonic sort [28] takes $O(N \log^2 N)$ time for N elements, SFD20 [7] take $O(N \log^2 N)$ time, where N is the sum of output size and input size. SMJ [4] has the same worst-case time complexity with SFD20.

D. Hash based Federated Binary Equi-Joins

Hash is also used to avoid unnecessary secure comparisons.

Main Idea. The main idea is to partition the tuples of one relation $\mathcal{R}$ into buckets that have the same hash value on the join attributes. Then, each tuple of the other relation $\mathcal{S}$ only needs to compare with a subset of tuples in $\mathcal{R}$ with the same hash value, which saves a large amount of secure comparisons.

Algorithm Sketch. Mohassel *et al.* [8] proposed an efficient algorithm (denoted as PPM20 by us due to the lack of algorithm name) based on the seminal hash join [26]. Specifically, when attribute values in the join condition are complex, each data owner locally performs randomized encodings for the attribute values in line 1. For secure operations, one data owner employs a cuckoo hash table [29] in a secret-shared form to guarantee that tuples in $\mathcal{R}$ with different join attributes appear in the same bucket with very low probability. The other data owner enumerates each tuple $s \in \mathcal{S}$ and securely retrieve all the tuples $r \in \mathcal{R}$ that have the same hash value with s, *i.e.*, Hash$[s.B]$ in lines 3-4. If the comparison between r and s is equal, we append CONCAT(r, s) to the join result Res. Otherwise, we append a dummy tuple to Res.

Remark. According to the complexity analysis in [8], both the time complexity and communication cost of PPM20 are $O(N)$, where N is the sum of input size and output size.

E. PSI based Federated Binary Equi-Joins

Beyond the previous categories that are commonly seen in the text books, one of the most popular and useful SMC primitives, Private Set Intersection (PSI), has also been leveraged to answer federated equi-joins.

Algorithm 3: Hash based method PPM20 [8]

Input: Two data owners with relations $\mathcal{R}$ and $\mathcal{S}$
Output: The join result $Res = \mathcal{R} \bowtie_{\mathcal{R}.A=\mathcal{S}.B} \mathcal{S}$

1 perform encodings for $\mathcal{R}.A$ and $\mathcal{S}.B$;
2 build a secure Cuckoo hash Hash based on $\mathcal{R}.A$;
3 **foreach** *tuple* s *in* $\mathcal{S}$ **do**
4 **foreach** *tuple* r *in* Hash$[s.B]$ **do**
5 **if** *attributes* $r.A$ *(securely) equal to* $s.B$ **then**
6 append CONCAT(r, s) to Res;
7 **else**
8 append a dummy tuple to Res;

Algorithm 4: PSI based method VOLE-PSI [10]

Input: Two data owners with relations $\mathcal{R}$ and $\mathcal{S}$
Output: The join result $Res = \mathcal{R} \bowtie_{\mathcal{R}.A=\mathcal{S}.B} \mathcal{S}$

1 primary keys X and payloads $PX \leftarrow$ encode $\mathcal{R}.A$;
2 primary keys Y and payloads $PY \leftarrow$ encode $\mathcal{R}.B$;
3 securely compute $Z \leftarrow X \cap Y$ by a PSI protocol;
4 $Res \leftarrow \{\text{CONCAT}(z, PX_z, PY_z) \mid \forall z \in Z\}$;

Main Idea. PSI allows two parties who hold the sets X and Y respectively, to compute their intersection $X \cap Y$ without revealing any information about the sets themselves. Accordingly, if the join condition only includes attributes with unique values (*e.g.*, primary keys X in $\mathcal{R}$ and Y in $\mathcal{S}$), the PSI result $X \cap Y$ will also indicate the joinable tuples.

Algorithm Sketch. As shown in Alg. 4, a primary-key-to-primary-key (PK-PK) federated binary equi-join is used to illustrate this algorithm family. Then, the sets X and Y represent collections of the people's names in the relation $\mathcal{R}$ and $\mathcal{S}$, respectively. The other attributes are usually called as the "payload" [10], and each element in the set is associated with a payload. To prepare the sets, each data owner first encodes their join attributes as the set elements and the corresponding payloads at the first step of our general framework. Then, a PSI protocol securely computes the intersection of both sets. Finally, the join result is generated by concatenating each intersected element and their payloads in X and Y.

Application Scope. PSI based methods are mainly used to answer primary-key-to-primary-key (PK-PK) federated equi-joins, and can hardly process join conditions with duplicated attribute values, such as the foreign keys. This is because the sets in PSI protocols usually assume no duplicated elements.

Remark. We mainly consider two circuit-based PSI protocols (OPPRF-PSI [22] and VOLE-PSI [10]) that encapsulate the intersection result in a secret-shared form. The OPPRF-PSI [22] has been applied in the federated equi-join algorithm SecYan [9]. By contrast, VOLE-PSI has the state-of-the-art performance. Both methods take linear time.

F. Extension to Federated Multi-way Equi-Joins

General Idea. The query processing of federated multi-way equi-joins is built upon the aforementioned algorithms. Specifically, based on the specific join order from the query coordinator, a federated multi-way equi-join is decomposed

TABLE I: Comparisons on federated equi-join algorithms (n is the input size and N is the sum of output size and input size).

Category	Algorithms	Time Complexity (for binary equi-joins)	#(Data Owners)	Equi-Join Type			Implementation	
				PK-PK	PK-FK	FK-FK	Original	Ours
Nested-loop based	SMCQL [1]	$O(n^2log^3n)$	2	✓	✓	✓	Java	C++
Nested-loop based	Conclave [6]	$O(n^2)$	3	✓	✓	✓	Python	C++
Nested-loop based	Shrinkwrap [5]	$O(n^2log^3n)$	m	✓	✓	✓	Unavailable	C++
Sort-merge based	SFD20 [7]	$O(Nlog^2N)$	2	✓	✓	✓	C++	C++
Sort-merge based	SMJ [4]	$O(Nlog^2N)$	m	✓	✓	✓	Unavailable	C++
Hash based	PPM20 [8]	$O(N)$	2	✓	✓	✓	Unavailable	C++
PSI based	SecYan [9]	$O(n)$	2	✓	✗	✗	C++	C++
PSI based	VOLE-PSI [10]	$O(n)$	2	✓	✗	✗	C++	C++

into multiple federated binary equi-joins. For each intermediate join, two relations are joined together to produce an intermediate result by using the aforementioned algorithm. Moreover, the intermediate result is often kept as secret-shares among the data owners to prevent information leakage. In the end, only the final join result is published to the query user.

Challenge: Excessive Dummy Tuples. In previous federated equi-join algorithms, padding dummy tuples in the join result is frequently used to guarantee the security. Take Alg. 1 [1] as an example. As shown in line 7, when the join selectivity is very low (*i.e.*, k is small), the intermediate result stored in the ORAM needs to be padded into $O(n^2)$ tuples, where n is the number of tuples in each relation. Similarly, when c tables are joined together, SMCQL needs to insert $O(n^c)$ dummy tuples in the worst case, which deteriorates the query efficiency.

Optimization: Shrink Dummy Tuples based on Differential Privacy. To eliminate redundant dummy tuples while still keeping the privacy, Bater *et al.* [5] proposed a resize mechanism in their algorithm Shrinkwrap under differential privacy (DP), which is the "de facto standard privacy notion" [30]. This DP mechanism first generates a truncated Laplacian noise η (*i.e.*, $\eta > 0$). Then, the number of dummy tuples can be shrunk from the size $O(n^2)$ to the size $k + \eta$, where n is the input size and k is the output size. This resize operation is recursively executed right after getting each intermediate result of federated binary equi-joins. Moreover, Bater *et al.* [5] also introduced a new cost model that indicated the overall I/O cost and could be helpful to determine the optimal join order.

G. Summary and Discussion

Table I presents an overall comparison of federated equi-join algorithms in terms of several aspects: the time complexity, number of data owners (m denotes an arbitrary number) in their original implementations, supported equi-join types and original implementations. For example, nested-loop based solutions usually have at least quadratic time complexity, while hash based solutions and PSI based solutions tend to have linear time complexity. As for equi-join types, PSI based solutions usually support join conditions that have involve primary keys (PK). Due to this reason, the time complexity $O(n)$ of the PSI based method is asymptotically equal to that ($O(N)$) of PPM20 [8] when answering PK-PK federated equi-joins, where n is the input size and N is the sum of output size and input size. Moreover, Table I demonstrates that there is no uniform implementation on the existing algorithms, and our benchmark FEJ-Bench aims to address this limitation.

TABLE II: Real datasets.

FK-FK Dataset	Slashdot	Jokes	Amazon
Input Size	2.5k×2.5k	15k×15k	120k×30k
#(Attributes)	2×2	2×2	2×2
FK-FK Result Size	3k	20k	100k
PK-FK/PK Dataset	IMDB-small	IMDB-medium	IMDB-large
Input Size	2.5k×1.5k	15k×5k	200k×10k
#(Attributes)	9×9	9×9	9×9
PK-FK Result Size	2.5k	15k	200k
PK-PK Result Size	1k	4k	10k

Although this paper primarily focuses on the federated equi-joins, existing work has studied other types of join queries over a data federation, such as theta-joins [1], [5], [6], join-aggregate queries [9], and spatial joins [18]. The idea of nested-loop based joins can be directly used to process these queries [1], [5], [6], [18]. The sort-merge based method, SMJ [4], can be also extended to process theta-joins, and the PSI based method SecYan [9] can be used to process join-aggregate queries. Since theta-joins are closely related to equi-joins, we have also evaluated the extensions of SMCQL [1], Conclave [6], and SMJ [4] to theta-joins. Due to the page limitation, please refer to our appendix [17] for more details.

IV. Experimental Setup

This section introduces the setup of our experimental study and the proposed benchmark FEJ-Bench.

A. Dataset

Our experiment study consists of five datasets that have been widely used in existing research, as shown in Table II.

- **Slashdot dataset** [31]. This dataset is collected by Slashdot, a technology news website with friend/foe links between users.
- **Jokes dataset** [32]. The Jokes dataset contains anonymous ratings of jokes by different users of the recommender system Jester [32] developed by UC Berkeley.
- **Amazon dataset** [16]. This dataset records the frequently co-purchased products on Amazon's website.
- **IMDB dataset** [33]. Two tables from the original IMDB dataset [33] are used: "title.basics" and "name.basics". The first table contains information about movies, such as their titles. The second table contains information about actors, such as their representative movies' names.
- **TPC-H dataset** [15]. The TPC-H dataset is a commonly-used benchmark dataset. We have used six tables in the TPC-H dataset: PART, PARTSUPP, CUSTOMER,

TABLE III: Parameter settings.

Query	Parameter	Setting
Federated Binary Equi-Join	Data Size	Left Table: 1k, 4k Right Table: 1k, 2k, 4k, 8k
	Join Selectivity	0.5k, 1k, 5k, 10k, 50k
	Data Skewness (Zipf factor)	0.1, 0.3, 0.5, 0.7, 0.9
Federated Multi-Way Equi-Join	Data Size	All: 0.05k, 0.1k, 0.2k, 0.5k
	Join Selectivity	0.05k, 0.1k, 0.2k, 0.5k

NATION, LINEITEM, and ORDERS. They have 9, 5, 8, 4, 16, and 9 attributes, respectively.

Data Partition. We consider two ways to determine the partitioned data of the above datasets in each owner.

- **Horizontal Partition** [34]. Since the Slashdot, Jokes, and Amazon datasets contain only one table, we randomly separate the tuples into two relations $\mathcal{R}$ and $\mathcal{S}$ that have the same schema but are held by different data owners. This kind of data federation is also known as a *horizontal data federation* in existing work [1], [5], [35].
- **Vertical Partition** [36]. By contrast, the IMDB and TPC-H datasets have more than one table. Thus, following a vertical partition in existing work [6], [9], each data owner holds a complete and different table, and these data owners consist a *vertical data federation*.

B. Query Workload

We follow existing work [13] to generate the query workload of federated equi-joins for each real-world dataset. Specifically, such a query retrieves indirectly connected pairs of users on the Slashdot dataset. This query can find users who rated the same jokes with the same rating on the Jokes dataset, and potential products that can be co-purchased on the Amazon dataset. As for the IMDB dataset, a federated equi-join query connects information on actors with their famous movies. For the TPC-H dataset, we modify the TPC-H query Q10 to generate the query workload. Moreover, as shown in Table II, the Slashdot, Jokes, and Amazon datasets only have FK-FK federated equi-joins, since their query attributes have duplicates. The IMDB dataset can support PK-FK and PK-PK federated equi-joins, while the TPC-H dataset can support all three equi-join types.

C. Parameter Settings

We evaluate the impact of several parameters, including the data size, join selectivity, and data skewness.

- For the *real datasets*, the only variable parameter is the data size. Thus, we divide the IMDB dataset into three different scales, and the scales of the Slashdot, Jokes, and Amazon datasets generally reflect different data sizes for answering FK-FK federated equi-joins.
- For the *TPC-H dataset*, we vary all the parameters based on its benchmark generator. For the federated multi-way equi-joins, we also vary the join orders.

As for the privacy parameters in Shrinkwrap, we follow the original paper [5] to set $\epsilon = 0.5$ and $\delta = 5 \times 10^{-5}$.

D. Evaluated Algorithms

Our benchmark includes all the existing algorithms that have been introduced in Sec. III, including *nested-loop* based methods [1], [5], [6], *sort-merge* based methods [4], [7], *hash* based method [8] and *PSI* based methods [9], [10]. Besides, following the existing work [1], we also include a plaintext baseline Plain-NLJ based on the nested-loop join.

To ensure a fair comparison, we implement all algorithms from scratch with a well-known SMC library ABY [23], and the symmetric security parameter are set to 128. These algorithms are programmed with C++ and compiled using GNU G++ 9.4.0. We leverage a data structure, called oblivious stack [11], to implement SMJ [4]. For the hashing based solution, PPM20 [8], we construct the hashing table by directly using the plaintext join keys instead of its randomized encoding, since the attributes in our datasets are relatively simple. Moreover, since PPM20 primarily considers PK-FK and PK-PK federated equi-joins only, we utilize the extension method in the Appendix A of [8] to make it also support FK-FK federated equi-joins.

E. Evaluation Metrics

We evaluate the performance of previous algorithms by three metrics: *running time, communication cost*, and *memory usage*. Here, the *running time* is the total time for getting the join result, the *communication cost* is the total size of data that are transferred via the network, and the *memory usage* denotes the peak main memory usage during the query processing. Following the settings in [18], [37], [38], the average results for answering 50 queries are reported.

F. Experimental Environment

We conduct the experiments on multiple machines, which act as the data owners of the data federation. Each machine is equipped with 24 2.40GHz Intel(R) Xeon(R) Gold 6240R CPU processors, 1TB main memory, and CentOS 7.9 OS.

V. Evaluation Results and Analysis

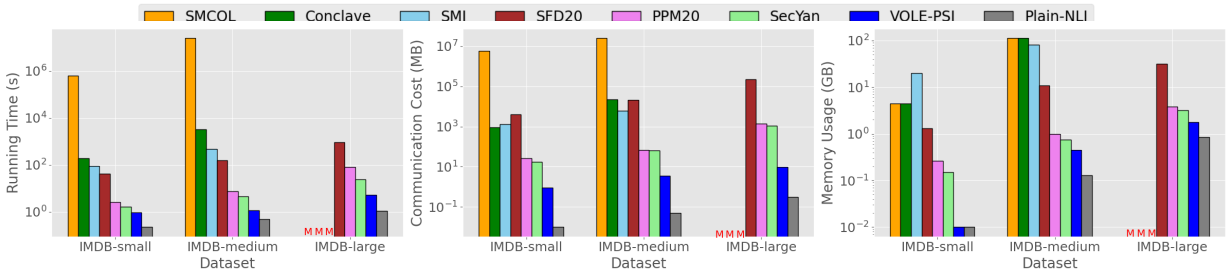
This section presents our experimental evaluations and analysis on existing federated equi-join algorithms.

A. Evaluation on Federated Binary Equi-Joins

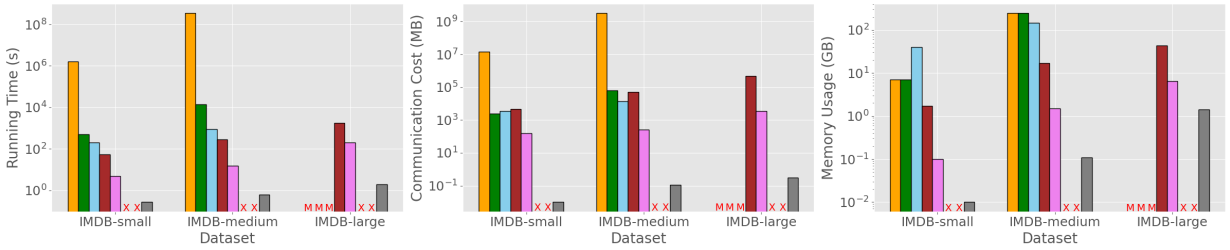
This subsection presents our evaluations and analysis on federated binary equi-joins when varying the equi-join types, data size, join selectivity, and data skewness. Notice that, since Shrinkwrap is designed for optimizing federated multi-way equi-joins and has the same result as SMCQL for federated binary equi-joins, Shrinkwrap is ignored in this subsection.

*1) **Results on Varying Query Types**:* Fig. 4 shows the results on real datasets with different equi-join types: Primary-Key-to-Primary-Key (PK-PK), Primary-Key-to-Foreign-Key (PK-FK), and Foreign-Key-to-Foreign-Key (FK-FK).

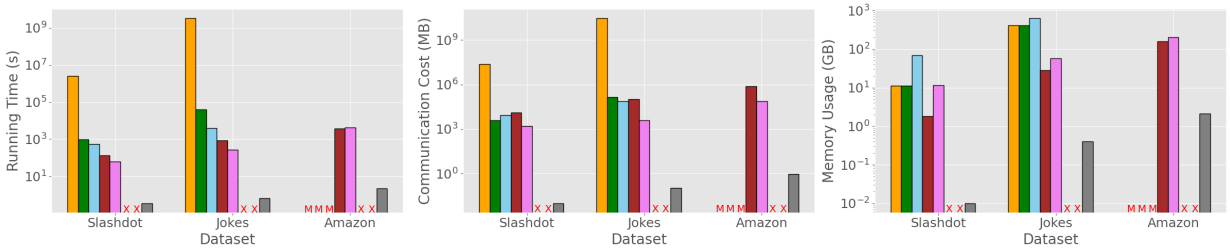
Primary Key Only: PK-PK Federated Equi-Joins. Fig. 4a presents the evaluations on the query type that involves primary keys over different scales of the real-world dataset IMDB. In terms of the *running time*, existing algorithms can be ranked from the fastest to the slowest as follows: VOLE-PSI, SecYan, PPM20, SFD20, SMJ, Conclave, and SMCQL.



(a) Results of Primary-Key-to-Primary-Key (PK-PK) federated binary equi-joins on real datasets



(b) Results of Primary-Key-to-Foreign-Key (PK-FK) federated binary equi-joins on real datasets



(c) Results of Foreign-Key-to-Foreign-Key (FK-FK) federated binary equi-joins on real datasets

Fig. 4: Evaluations on different query types of federated binary equi-joins on real datasets. "X" denotes that an algorithm does not support this query type. "M" denotes that an algorithm is crashed due to the usage of excessive memory.

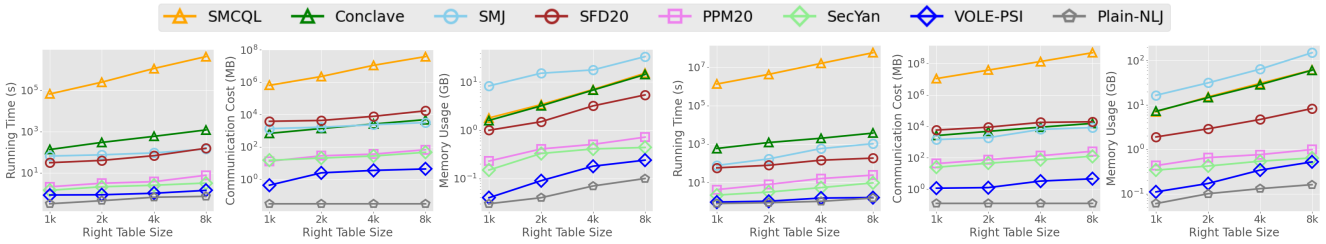
Among these algorithms, Conclave and SMCQL are notably slower than the others, since they both need a large number ($O(mn)$) of secure comparisons that take high time cost. The improvement of VOLE-PSI and SecYan over the others demonstrate that PSI is effective to alleviate the time cost lead by secure computations. Moreover, all the secure algorithms can be significantly slower than plaintext baseline Plain-NLJ by at least $4.6\times$. As for the *communication cost*, the performance of nested-loop based methods is worse than the other three algorithm categories, and the PSI based methods tend to take lower communication than the others. The total communication cost of existing algorithms can be at least $31.5\times$ higher than that of Plain-NLJ. In terms of the *memory usage*, SMJ, Conclave, and SMCQL are crashed due to the usage of excessive memory, since they have to allocate a large memory space for the join result to ensure obliviousness. By contrast, the other algorithm does not suffer from the high memory cost for this query type.

Foreign Key Involved: PK-FK & FK-FK Federated Equi-Joins. Fig. 4b and Fig. 4c illustrate the experimental results of PK-FK and FK-FK federated equi-Joins on real datasets, respectively. Since VOLE-PSI and SecYan do not support these query types, we do not report the results. Among the other secure algorithms, PPM20 is the fastest in the metric of *running time*, but still can be up to 2 orders of magnitude slower than the plaintext equi-join Plain-NLJ. SMJ is the runner-up in the running time, while the other secure algorithms consume too much main memory space ($>$ 1TB) to be terminated when meeting the large data size (*i.e.*, IMDB-large and Amazon datasets). The result patterns of the *communication cost* are similar to those of PK-PK federated equi-join. For instance, PPM20 has the lowest communication cost, while SMCQL takes the highest. In terms of the *memory usage*, SMJ is the lowest, which still requires over 150GB main memory space on the Amazon dataset.

Comparisons Across Different Query Types. As illustrated in Fig. 4, existing federated equi-join algorithms are more efficient to process the PK-FK type than the FK-FK type by taking 1.5-10.8$\times$ shorter time and 1.6-14.3$\times$ lower communication. One possible reason is that foreign keys involve duplicated attribute values that tend take extra cost in the query processing. The comparisons across different equi-join query types also indicate that FK-FK federated equi-join may be even more challenging than the other two types.

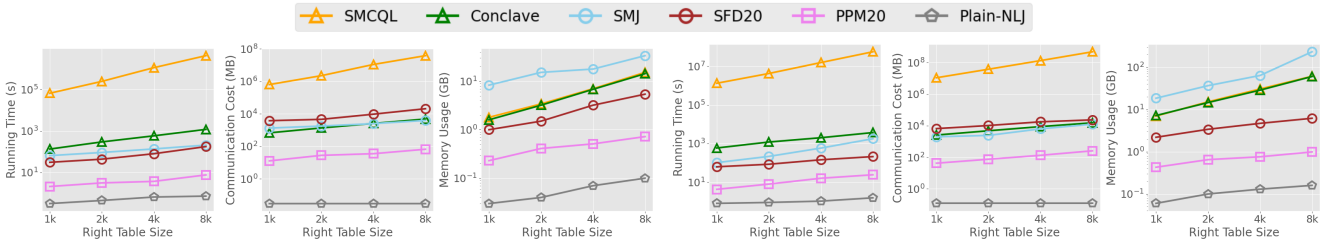
*2) **Results on Varying Data Size**:* In Fig. 5 and Fig. 6, we present an evaluation on different sizes of the left relation $\mathcal{R}$ and right relation $\mathcal{S}$. Due to the page limitation, please refer to our full paper [17] for the results of FK-FK federated binary equi-joins, which have similar patterns with Fig. 6.

Increase Right Table Size. When the right table size $|\mathcal{S}|$ increases by $8\times$, the running time and communication cost of secure algorithms queries notably increase by 3.6-64$\times$ and 5.1-60$\times$ on PK-FK queries. Similarly, the metrics increase 1.7-64$\times$ and 3.1-59$\times$ on PK-PK queries. The efficiency of SMCQL is much more sensitive to $|\mathcal{S}|$ than the others, which



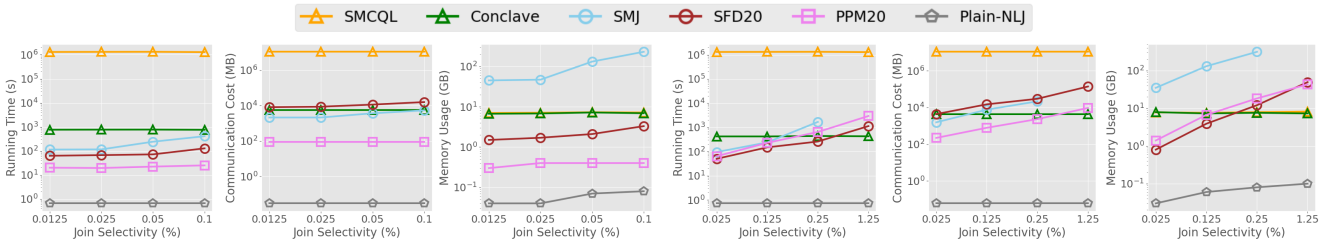
(a) Results when the left table size is 1k (b) Results when the left table size is 4k

Fig. 5: Evaluation on varying input table sizes of PK-PK federated binary equi-joins on the TPC-H dataset.



(a) Results when the left table size is 1k (b) Results when the left table size is 4k

Fig. 6: Evaluation on varying input table sizes of PK-FK federated binary equi-joins on the TPC-H dataset.



(a) Varying join selectivity (PK-FK federated binary equi-joins) (b) Varying join selectivity (FK-FK federated binary equi-joins)

Fig. 7: Evaluation on varying join selectivity of federated binary equi-joins on the TPC-H dataset.

is aligned with its high complexity in Table I. By contrast, VOLE-PSI, SecYan, SFD20 and PPM20 are relatively less sensitive to the table size. Moreover, PPM20 is always the most efficient solution in PK-FK join, while VOLE-PSI is always the most efficient one in PK-PK join.

Increase Left Table Size. By comparing the results between Fig. 6a and Fig. 6b, we can observe federated equi-join algorithms are also sensitive to the left table size $|\mathcal{R}|$. For the secure algorithms, their degrees of sensitivities to $|\mathcal{R}|$ are generally similar to those to $|\mathcal{S}|$.

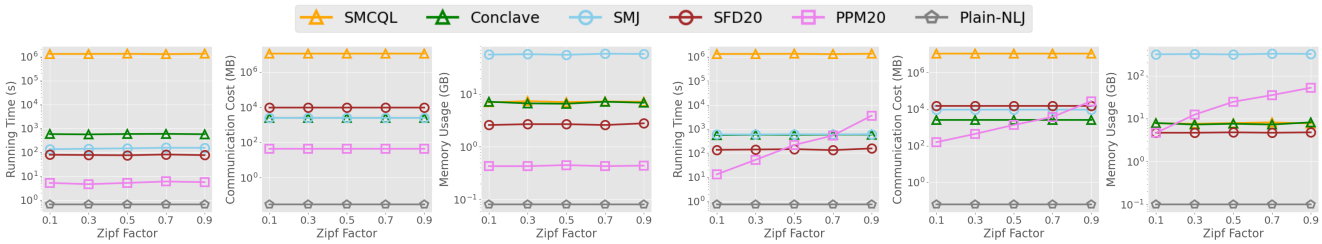
Increase Both Table Sizes. When increase sizes of both tables from $1k$ to $4k$ in the PK-FK join, the running time of SMCQL, Conclave, SMJ, SFD20, and PPM20 increases by 237, 15, 5, 9, 8×, respectively. Their communication cost and memory cost also get higher by 5-210× and 3-17×, respectively. By contrast, the running time of the plaintext baseline Plain-NLJ only increases by 30%. This pattern proves that federated equi-joins are much more sensitive to the dataset scalability than plaintext equi-joins.

*3) **Results on Varying Join Selectivity**:* Fig. 7 shows the experimental results of PK-FK and FK-FK federated equi-joins when the join selectivity increases from 0.0125% to 1.25%. Due to the page limitation, please refer to [17] for our results of PK-PK federated binary equi-joins.

PK-FK Federated Equi-Joins. As shown in Fig. 7a, only SFD20 and SMJ show a slightly upward trend on the running time and communication cost when the join selectivity goes up. As for the communication cost, all algorithms remain relatively stable. In terms of memory usage, SFD20 still shows a slightly rise trend, while SMJ requires up to 5× more space when the join selectivity raises from 0.0125% to 1.25%.

FK-FK Federated Equi-Joins. As shown in Fig. 7b, all the federated equi-join algorithms except for SMCQL and Conclave show obvious changes. Specifically, SMJ, SFD20, and PPM20 take 22-90× longer running time and 21-72× higher communication cost when the join selectivity increases from 0.025% to 1.25%. Meanwhile, Conclave transforms from being the fourth most efficient method to becoming the most efficient one among all secure algorithms. Besides, by comparing the changes in Fig. 7a and Fig. 7b, we also observe that the efficiency of existing algorithms are more likely to be affected by the selectivity of FK-FK federated equi-joins than PK-FK federated equi-joins.

*4) **Results on Varying Data Skewness**:* The data skewness of the TPC-H dataset is usually varied by changing the Zipf factor to adjust the foreign keys [12]. Thus, the primary keys (PK) are not affected, and we only evaluate PK-FK and FK-FK federated equi-joins under this parameter setting.



(a) Results of varying data skewness (PK-FK equi-joins)

(b) Results of varying data skewness (FK-FK equi-joins)

Fig. 8: Evaluation on varying data skewness of federated binary equi-joins on the TPC-H dataset.

TABLE IV: Ranking of federated binary equi-join algorithms in the query processing time.

Federated Equi-Join Types	Join Selectivity	Ranking by Running Time
Primary-Key-to-Primary-Key (PK-PK)	Any	VOLE-PSI < SecYan < PPM20 < SFD20 < SMJ < Conclave < SMCQL
Primary-Key-to-Foreign-Key (PK-FK)	Any	PPM20 < SFD20 < SMJ < Conclave < SMCQL
Foreign-Key-to-Foreign-Key (FK-FK)	Low	PPM20 < SFD20 < SMJ < Conclave < SMCQL
	High	Conclave < PPM20 < SFD20 < SMJ < SMCQL

TABLE V: Ranking of federated binary equi-join algorithms in the total communication cost.

Federated Equi-Join Types	Join Selectivity	Ranking by Communication Cost
Primary-Key-to-Primary-Key (PK-PK)	Any	VOLE-PSI < SecYan < PPM20 < SMJ < Conclave < SFD20 < SMCQL
Primary-Key-to-Foreign-Key (PK-FK)	Any	PPM20 < SMJ < Conclave < SFD20 < SMCQL
Foreign-Key-to-Foreign-Key (FK-FK)	Low	PPM20 < SMJ < Conclave < SFD20 < SMCQL
	High	Conclave < PPM20 < SMJ < SFD20 < SMCQL

PK-FK Federated Equi-Joins. As shown in Fig. 8b, all the algorithms remain a relatively stable performance in terms of running time, communication cost, and memory usage. This is because the primary keys are unchanged such that the overall join result size (*i.e.*, join selectivity) remains stable.

FK-FK Federated Equi-Joins. However, when conducting this experiment on FK-FK federated equi-joins, hashing based solution PPM20 shows an upward trend when the data skewness increases. This might because the underlying hash functions encounter more collisions when data gets skewed and eventually leads to extra time cost. For instance, when the Zipf factor varies from 0.1 to 0.9, the rank of PPM20 in any of these metrics (from the most efficient to the least efficient) decreases from the first to the fourth. The pattern indicates that hashing based solutions are sensitive to data skewness for FK-FK federated equi-joins. By contrast, the other algorithms are not influenced by the data skewness.

*5) **Experimental Observations and Analysis**:* Based on the previous experimental results, we present our overall rankings of existing federated equi-join algorithms in Table IV-VI. The order is based on the average rank of these algorithms in previous experiments. Based on these experimental observations, we have made the following in-depth analysis.

Observation #1: in most cases, the ranking by running time or communication is "PSI < hash < sort-merge < nested-loop". As shown in Table IV and Table V, the hash based method PPM20 takes shorter time and communication cost than the sort-merge based solutions (SFD20 and SMJ), while at least one sort-merge based method often performs better than the nested-loop based solutions (SMCQL and Conclave). Moreover, when answering PK-PK equi-joins, the PSI based solutions VOLE-PSI and SecYan are always more efficient than the others.

Observation #2: join selectivity can have a notable impact on the query efficiency for most algorithms. Both the time and communication cost of *hash* based and *sort-merge* based methods notably increase as the join selectivity rises. Consequently, the overall ranking for FK-FK federated equi-joins can be changed when encountering a high join selectivity. As shown in Table IV and V, the *nested-loop* based method Conclave becomes the most efficient in this scenario.

Observation #3: main memory usage can be also an overlooked efficiency bottleneck in existing work. As shown in Table VII, on the Amazon dataset, all secure algorithms consumed hundreds of gigabytes of main memory and some (SMJ, Conclave, and SMCQL) even exceeded 1TB, while the plaintext method Plain-NLJ took only 2GB main memory. In contrast, when answering PK-PK federated equi-joins, the memory usages of *PSI* based solutions (VOLE-PSI and SecYan) and *hash* based method (PPM20) are comparably normal. The results indicate that a large memory cost may need to be paid for federated equi-joins over foreign keys.

B. Evaluation on Federated Multi-way Equi-Joins

This subsection evaluates the federated multi-way equi-joins when varying the data size and join orders. We exclude VOLE-PSI and SecYan from this evaluation, since a federated multi-way equi-join easily involves foreign keys in the join condition, which cannot be supported by PSI based methods.

*1) **Results on Varying Data Size**:* Fig. 9 presents the experimental results when varying each table size at different levels of the join selectivity.

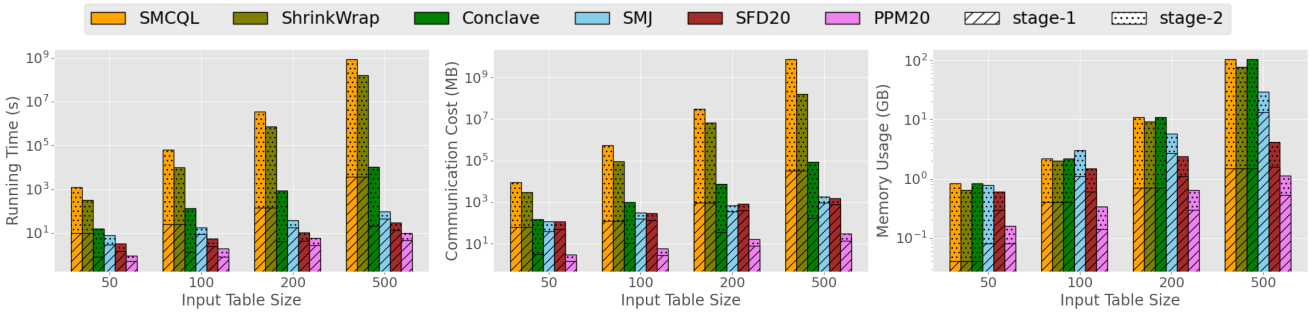
Results When Join Selectivity is Low. As shown in Fig. 9a, when each table size increases from 50 to 500 at low join se-

TABLE VI: Ranking of federated binary equi-join algorithms in the main memory usage.

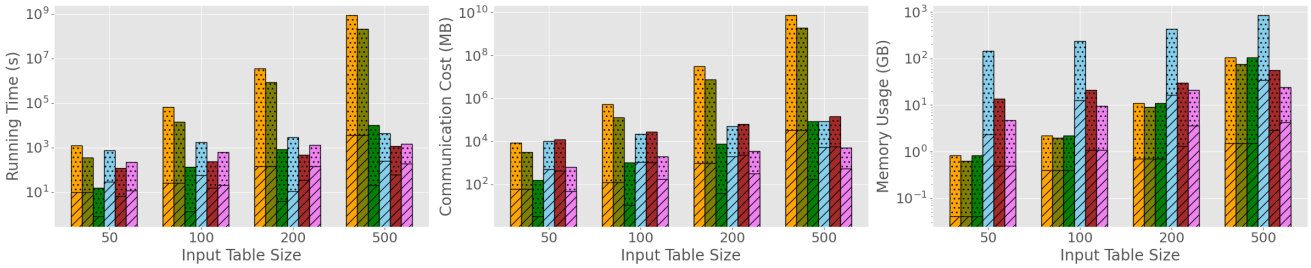
Federated Equi-Join Types	Join Selectivity	Ranking by Memory Usage
Primary-Key-to-Primary-Key (PK-PK)	Any	VOLE-PSI <SecYan < PPM20 < SFD20 < SMJ < Conclave = SMCQL
Primary-Key-to-Foreign-Key (PK-FK)	Any	PPM20 < SFD20 < SMJ < Conclave = SMCQL
Foreign-Key-to-Foreign-Key (FK-FK)	Low	SFD20 < PPM20 < Conclave = SMCQL < SMJ
	High	Conclave = SMCQL < PPM20 < SFD20 < SMJ

TABLE VII: Maximum main memory usage for federated binary equi-joins.

Dataset	Relation Size	Equi-Join Type	Plain-NLJ	VOLE-PSI	SecYan	PPM20	SFD20	SMJ	Conclave	SMCQL
IMDB-large	200k×10k	PK-PK	0.8GB	1.8GB	3GB	4GB	32GB	238GB	>1TB	>1TB
IMDB-large	200k×10k	PK-FK	1GB	N/A	N/A	7GB	44GB	368GB	>1TB	>1TB
Amazon	120k×30k	FK-FK	2GB	N/A	N/A	201GB	158GB	>1TB	>1TB	>1TB



(a) Experimental result at log-scale when the join selectivity is relatively low (*i.e.*, 0.01%)



(b) Experimental result at log-scale when the join selectivity is relatively high (*i.e.*, 1%)

Fig. 9: Evaluation on varying each table size of federated multi-way equi-joins on the TPC-H dataset.

lectivity, PPM20 still has the shortest running time and lowest communication, followed by SFD20, SMJ and Conclave. By contrast, SMCQL and Shrinkwrap are notably less efficient than the others. Moreover, when each relation size is expanded by 10×, the running time of PPM20, SFD20, SMJ, and Conclave has increased by 10.8, 9.2, 12.6, 651.2×, respectively. The other algorithms, SMCQL and Shrinkwrap take 2-5 orders of magnitude longer running time, which is more sensitive to data size than the aforementioned methods. This is because SMCQL need $O(n^2 log^3 n)$ secure computations, and Shrinkwrap needs fewer secure computations than it due to the usage of differential privacy in truncating the cardinality of intermediate results.

Results When Join Selectivity is High. By comparing the results in Fig. 9a with those in Fig. 9b, we observe the runtime of PPM20, SFD20, and SMJ increases by up to 339, 99, 45× respectively, when the join selectivity gets high. At the same time, their communication cost increases by up to 327, 106, 73×, respectively. By contrast, the changes of Conclave, Shrinkwrap, and SMCQL are relatively stable. Moreover, we can also observe that the main memory usage of all the algorithms increases with the expansion of the input data sizes. Eventually, when there are 500 tuples in each relation, PPM20 takes the lowest memory cost.

Result Breakdown. We also use different hatching patterns on the bars to indicate the breakdowns at different stages of a federated multi-way equi-join over three tables. The bottom part, which is filled with slashes, indicates the cost when joining the first two tables (*i.e.*, stage 1). The upper part, which is filled with dots, indicates the cost when joining the last table with the intermediate results of the first two tables (*i.e.*, stage 2). Based on the breakdowns, we can observe that for SMCQL, Shrinkwrap, and Conclave, most of the running time is consumed during the second stage. This is because the input in the second stage involves the Cartesian product of two input tables, which is much larger than the input in the first stage. Moreover, in Fig. 9a, the time, communication, and memory costs of PPM20, SFD20, and SMJ at the second stage are close to those at their first stage. However, as the join selectivity increases, the overall costs of these three algorithms at the second stage significantly surpass those at the first stage. This difference arises due to the larger size of intermediate

TABLE VIII: Ranking of federated multi-way equi-join algorithms in the running time.

Join Selectivity	Ranking by Running Time
Low	PPM20 < SFD20 < SMJ < Conclave < Shrinkwrap < SMCQL
High	SFD20 < Conclave < PPM20 < SMJ < Shrinkwrap < SMCQL

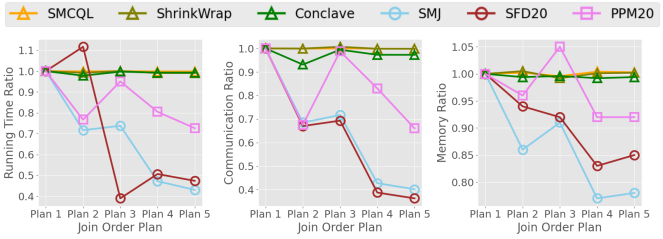


Fig. 10: Evaluation on varying join orders of federated multi-way equi-joins on the TPC-H dataset.

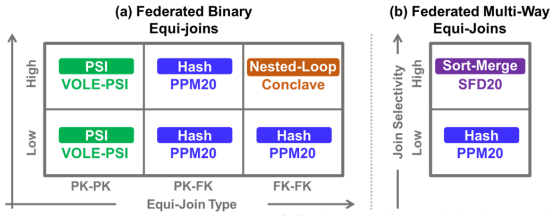


Fig. 11: Recommendations of federated equi-join methods.

results in Fig. 9b compared to that in Fig. 9a.

*2) **Results on Varying Join Orders**:* We modify the query template Q10 in the TPC-H benchmark to test the impact of join orders as follows.

```
SELECT l_partkey, l_orderkey, o_custkey,
       c_nationkey, n_regionkey
FROM   LINEITEM, ORDERS, CUSTOMER, NATION
WHERE  l_orderkey = o_orderkey and
       c_custkey = o_custkey and
       c_nationkey = n_nationkey
```

Parameters. The relation sizes are set to be $|\text{LINEITEM}| = 400$, $|\text{ORDERS}| = 100$, $|\text{CUSTOMER}| = 10$, $|\text{NATION}| = 25$. The data size is set to be small, since some methods have already taken over 50 hours and we tend to get results for all methods. Based on the relational algebra expression, this query has five different join orders as follows.

$$\text{Plan 1}: \big((\text{LINEITEM} \bowtie \text{ORDERS}) \bowtie \text{CUSTOMER}\big) \bowtie \text{NATION}$$
$$\text{Plan 2}: (\text{LINEITEM} \bowtie \text{ORDERS}) \bowtie (\text{CUSTOMER} \bowtie \text{NATION})$$
$$\text{Plan 3}: \big(\text{LINEITEM} \bowtie (\text{ORDERS} \bowtie \text{CUSTOMER})\big) \bowtie \text{NATION}$$
$$\text{Plan 4}: \text{LINEITEM} \bowtie \big((\text{ORDERS} \bowtie \text{CUSTOMER}) \bowtie \text{NATION}\big)$$
$$\text{Plan 5}: \text{LINEITEM} \bowtie \big(\text{ORDERS} \bowtie (\text{CUSTOMER} \bowtie \text{NATION})\big)$$

Results. Since the results of the compared algorithms lie in quite different scales, we use the ratio of changes over the result of the first join order to demonstrate the impact of different join orders. As shown in Fig. 10, only SMCQL and Shrinkwrap remain stable when varying the join orders. The running time and communication cost of all the others are obviously affected by the join orders, including Conclave that is tested to remain stable when varying several parameters of federated binary equi-joins. Moreover, we can also observe that different methods have different optimal join orders. For example, the running time of PPM20, SFD20 and SMJ can be reduced by 27%, 61% and 57% respectively, when replacing with their optimal join orders.

*3) **Experimental Observations and Analysis**:* Because federated binary equi-joins are essentially building blocks of federated multi-way equi-joins, we expect that the major observations from federated binary equi-joins would generally remain for federated multi-way equi-joins (*i.e.*, the following observations #1-#3). Moreover, we have also summarized our observation from the evaluation of different join orders.

Observation #1: when the join selectivity is low, the ranking by running time or communication cost still follows the order of "hash < sort-merge < nested-loop". Table VIII and Table IX present the performance ranking of all the algorithms based on their running time and communication cost. When the join selectivity is low, the *hash* based method PPM20 has the shortest running time and the lowest communication cost, with the *sort-merge* based method SFD20 as the runner-up.

Observation #2: the join selectivity continues to impact query efficiency. When the join selectivity is high, the running time of the *hash* based method PPM20 exceeds that of the *sort-merge* based method SFD20 and *nested-loop* based method Conclave. Additionally, the communication cost of the *nested-loop* based method Conclave is lower than that of the *sort-merge* based methods, SFD20 and SMJ.

Observation #3: the memory usage of existing methods remains an important consideration. As depicted in Table X, we present the maximum main memory usage during the query processing of federated multi-way equi-joins over three tables, each with only 500 tuples. Notably, when the join selectivity is high, the memory usage can exceed 19GB. Even when the join selectivity gets low, the space cost of any algorithm is over 0.6GB, which is much higher than the raw data size (21KB).

Observation #4: the efficiency of most algorithms is easily affected by join orders and the optimal join order varies across different methods. In fact, our evaluation resolves contradicted ideas in existing research on federated multi-way equi-joins. For instance, some existing work [9] assumed "*all the cost-based query optimization techniques useless*" in processing a federated equi-joins. However, others argue that "*one potential solution for scaling* SMCQL $\cdots$ *would require an analytical cost model*" [1]. Since join order optimization is a typical cost-based query optimization technique [26], our results demonstrate that this technique (1) remains *meaningful* for *hash* based and *sort-merge* based methods, and (2) may be *relatively meaningless* for *nested-loop* based methods.

C. Experimental Takeaways

The lessons we have learned are summarized as follows.

- In most cases, **the overall efficiency ranking from best to worst is "PSI > hash > sort-merge > nested-loop"**. Based on the pros/cons of these evaluated algorithms, we provide our recommendations for the best solution to federated equi-joins under different scenarios in Fig. 11.

TABLE IX: Ranking of federated multi-way equi-join algorithms in the communication cost.

Join Selectivity	Ranking by Communication Cost
Low	PPM20 < SFD20 < SMJ < Conclave < Shrinkwrap < SMCQL
High	PPM20 < Conclave < SMJ < SFD20 < Shrinkwrap < SMCQL

TABLE X: Maximum main memory usage for federated multi-way equi-joins.

Dataset	Each Relation Size	Join Selectivity	PPM20	SFD20	SMJ	Conclave	Shrinkwrap	SMCQL
TPC-H	500	Low	0.6GB	2.6GB	16.5GB	102.5GB	75.4GB	102.5GB
TPC-H	500	High	19.8GB	52.7GB	813.3GB	102.5GB	75.4GB	102.5GB

- **High join selectivity can change the overall efficiency ranking**. Either the *nested-loop* based method Conclave or the *sort-merge* based method SFD20 can become the most efficient when the join selectivity is high.
- Since **an optimal join order can significantly improve the query efficiency of most efficient algorithms**, it can be a research direction that deserves more attention.
- **Memory usage can potentially be a performance bottleneck even when the dataset gets large**. For example, when two relations have $120k \times 30k$ tuples on the Amazon dataset, the evaluated methods require over 150GB of main memory, while the plaintext method takes 2GB.

VI. Future Direction

In this section, we introduce the challenges and opportunities in the future study of federated equi-joins.

Boosting Scalability and Robustness. It is important to further boost the scalability and robustness, because (1) existing federated equi-join algorithms are much less scalable than the plaintext nested-loop equi-join, and (2) most of these methods are easily affected by factors like join selectivity. One potential direction is to use dedicated hardware, such as Trusted Execution Environments (TEEs) [11]. Maliszewski *et al.* [33] conducted an evaluation study on how plaintext binary equi-join algorithms will perform in TEEs. Although their problem setting is different from us, their primary results demonstrate that rethinking existing federated equi-join algorithms from the perspective of the hardware-software co-design is important to boost the scalability by a large margin.

Malicious Security. At this present, most of the existing solutions to federated equi-joins assume the attackers are *semi-honest*. In other words, each data owner and the query coordinator will strictly follow the pre-specified query processing procedure and try to infer the sensitive information during the computations across multiple parties. By contrast, another type of attackers, *i.e.*, *malicious attackers*, can corrupt the other parities in the data federation. Moreover, the corrupted parties can also deviate from the pre-specified query processing procedure, *e.g.*, changing the input or output of the intermediate computations. For example, when answering federated binary equi-joins in our Example 1 with an extra condition "Disease = Flu" in the predicate , a malicious attacker can corrupt the data owner the table $\mathcal{S}$ and make him input the records with "Disease = Cancer" to infer more information from the table $\mathcal{R}$. Existing work [21] has started to explore this problem, but it is still open to design a scalable solution that can overcome more than 1 corrupted party.

Federated Join Order Optimization. When dealing with federated multi-way equi-joins, the selection of the optimal join order has been demonstrated to be crucial to the query efficiency in our experiments. Existing solutions to the join order optimization problem usually assume that the underlying data is public, and hence cannot be directly used in our problem setting. By contrast, a data federation usually considers the private/sensitive data that is distributed among several data owners. Thus, to determine the optimal join order, it is important and challenging to securely build accurate cost models (*i.e.*, an estimation of the query latency) without leaking sensitive information about the private datasets.

VII. Conclusion

In this paper, we conduct a comprehensive experimental study on federated equi-joins, and introduce the first benchmark called FEJ-Bench for this type of queries. This evaluation study comprises five datasets from diverse application scenarios and eight state-of-the-art methods, all implemented with a uniform programming framework by us. We also conduct standardized and comprehensive comparisons on these methods under various parameters. The experimental results reveal the pros/cons of these methods, indicating that no single algorithm can completely dominate the others in terms of the query efficiency. Thus, we have also made our benchmark publicly available, and expect that it will enrich future research for more researchers and practitioners on data federation.

Acknowledgments

This work is partially supported by National Science Foundation of China (NSFC) under Grant No. U21A20516, 6233000216, and 62076017, Beijing Natural Science Foundation No. Z230001, CCF-Huawei Populus Grove Fund, Didi Collaborative Research Program NO2231122-00047, and the Beihang University Basic Research Funding No. YWF-22-L-531. Zimu Zhou's research is supported by Chow Sang Sang Group Research Fund No. 9229139.

References

[1] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers, "SMCQL: secure query processing for private data networks," *PVLDB*, vol. 10, no. 6, pp. 673–684, 2017.

[2] A. Bharadwaj and G. Cormode, "An introduction to federated computation," in *SIGMOD*, 2022, pp. 2448–2451.

[3] Y. Tong, Y. Zeng, Z. Zhou, B. Liu, Y. Shi, S. Li, K. Xu, and W. Lv, "Federated computing: Query, learning, and beyond," *IEEE Data Eng. Bull.*, vol. 46, no. 1, pp. 9–26, 2023.

[4] Z. Chang, D. Xie, S. Wang, and F. Li, "Towards practical oblivious join," in *SIGMOD*, 2022, pp. 803–817.

[5] J. Bater, X. He, W. Ehrich, A. Machanavajjhala, and J. Rogers, "Shrinkwrap: Efficient SQL query processing in differentially private data federations," *PVLDB*, vol. 12, no. 3, pp. 307–320, 2018.

[6] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros, "Conclave: secure multi-party computation on big data," in *EuroSys*, 2019, pp. 3:1–3:18.

[7] S. Krastnikov, F. Kerschbaum, and D. Stebila, "Efficient oblivious database joins," *PVLDB*, vol. 13, no. 11, pp. 2132–2145, 2020.

[8] P. Mohassel, P. Rindal, and M. Rosulek, "Fast database joins and PSI for secret shared data," in *CCS*, 2020, pp. 1271–1287.

[9] Y. Wang and K. Yi, "Secure yannakakis: Join-aggregate queries over private data," in *SIGMOD*, 2021, pp. 1969–1981.

[10] S. Raghuraman and P. Rindal, "Blazing fast PSI from improved OKVS and subfield VOLE," in *CCS*, 2022, pp. 2505–2517.

[11] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018.

[12] S. Schuh, X. Chen, and J. Dittrich, "An experimental comparison of thirteen relational equi-joins in main memory," in *SIGMOD*, 2016, pp. 1961–1976.

[13] Z. Huang and S. Chen, "Density-optimized intersection-free mapping and matrix multiplication for join-project operations," *PVLDB*, vol. 15, no. 10, pp. 2244–2256, 2022.

[14] S. Kumar, D. E. Culler, and R. A. Popa, "MAGE: nearly zero-cost virtual memory for secure computation," in *OSDI*, 2021, pp. 367–385.

[15] TPC-H, 2023. [Online]. Available: https://www.tpc.org/tpch/

[16] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The dynamics of viral marketing," *ACM Trans. Web*, vol. 1, no. 1, p. 5, 2007.

[17] "FEJ-Bench: A benchmark for federated equi-joins," 2023. [Online]. Available: https://github.com/BUAA-BDA/Hufu-FedJoin-Benchmark

[18] Y. Tong, X. Pan, Y. Zeng, Y. Shi, C. Xue, Z. Zhou, X. Zhang, L. Chen, Y. Xu, K. Xu, and W. Lv, "Hu-fu: Efficient and secure spatial queries over data federation," *PVLDB*, vol. 15, no. 6, pp. 1159–1172, 2022.

[19] P. Jurczyk, L. Xiong, and S. Goryczka, "Dobjects+: Enabling privacy-preserving data federation services," in *ICDE*, 2012, pp. 1325–1328.

[20] R. Poddar, S. Kalra, A. Yanai, R. Deng, R. A. Popa, and J. M. Hellerstein, "Senate: A maliciously-secure MPC platform for collaborative analytics," in *30th USENIX Security Symposium,*, 2021, pp. 2129–2146.

[21] F. Han, L. Zhang, H. Feng, W. Liu, and X. Li, "Scape: Scalable collaborative analytics system on private database with malicious security," in *ICDE*, 2022, pp. 1740–1753.

[22] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient circuit-based PSI with linear communication," in *EUROCRYPT*, 2019, pp. 122–153.

[23] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *NDSS*, 2015.

[24] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "Oblivm: A programming framework for secure computation," in *S&P*, 2015, pp. 359–376.

[25] Obliv-C, 2023. [Online]. Available: https://oblivc.org/

[26] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts, Seventh Edition*. McGraw-Hill Book Company, 2020.

[27] Y. Li and M. Chen, "Privacy preserving joins," in *ICDE*, 2008, pp. 1352–1354.

[28] K. E. Batcher, "Sorting networks and their applications," in *AFIPS*, 1968, pp. 307–314.

[29] D. Demmler, P. Rindal, M. Rosulek, and N. Trieu, "PIR-PSI: scaling private contact discovery," *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 4, pp. 159–178, 2018.

[30] N. Li, M. Lyu, D. Su, and W. Yang, *Differential Privacy: From Theory to Practice*. Morgan & Claypool Publishers, 2016.

[31] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Math.*, vol. 6, no. 1, pp. 29–123, 2009.

[32] K. Y. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Inf. Retr.*, vol. 4, no. 2, pp. 133–151, 2001.

[33] K. J. Maliszewski, J. Quiané-Ruiz, J. Traub, and V. Markl, "What is the price for joining securely? benchmarking equi-joins in trusted execution environments," *PVLDB*, vol. 15, no. 3, pp. 659–672, 2021.

[34] M. Kantarcioglu and C. Clifton, "Privacy-preserving distributed mining of association rules on horizontally partitioned data," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1026–1037, 2004.

[35] Y. Shi, Y. Tong, Y. Zeng, Z. Zhou, B. Ding, and L. Chen, "Efficient approximate range aggregation over large-scale spatial data federation," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 418–430, 2023.

[36] J. Vaidya and C. Clifton, "Privacy preserving association rule mining in vertically partitioned data," in *SIGKDD*, 2002, pp. 639–644.

[37] Y. Tong, L. Chen, and B. Ding, "Discovering threshold-based frequent closed itemsets over probabilistic data," in *ICDE*, 2012, pp. 270–281.

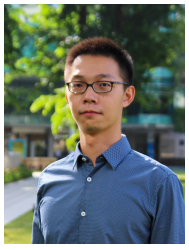
[38] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *ICDE*, 2016, pp. 49–60.

Shuyuan Li is currently working toward the Ph.D. degree in the School of Computer Science and Engineering, Beihang University. Her major research interests include federated data management, privacy-preserving data analytics, etc.

Yuxiang Zeng received the Ph.D. degree in computer science and engineering from the Department of Computer Science and Engineering, the Hong Kong University of Science and Technology, in 2022. He is currently an associate professor in the School of Computer Science and Engineering, Beihang University. His research interests include spatio-temporal data management, federated data management, privacy-preserving data analytics, crowdsourcing, etc.

Yuxiang Wang is currently working toward the Master degree in the School of Computer Science and Engineering, Beihang University. His major research interests include federated data management, privacy-preserving data analytics, etc.

Yiman Zhong is currently working toward the Master degree in the School of Computer Science and Engineering, Beihang University. Her major research interests include federated data management, privacy-preserving data analytics, etc.



Zimu Zhou received the B.E. degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2011, and the Ph.D. degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, in 2015. He is currently an assistant professor at the School of Data Science, City University of Hong Kong. His research focuses on mobile and ubiquitous computing. He is a member of the IEEE.

Yongxin Tong received the Ph.D. degree in computer science and engineering from the Hong Kong University of Science and Technology in 2014. He is currently a professor in the School of Computer Science and Engineering, Beihang University. His research interests include federated learning, federated data management, spatio-temporal data analytics, privacy-preserving data analytics, crowdsourcing, uncertain data management, etc. He is a member of the IEEE.