Efficient and Secure Multiparty Querying over Federated Graph Databases

Nouf Aljuaid[1,2], Alexei Lisitsa[2] and Sven Schewe[2]
[1]*Department of Information Technology, Taif University, Saudi Arabia*
[2]*Department of Computer Science, University of Liverpool, Liverpool, U.K.*

Keywords: Graph Databases, SMPC, Federated Databases, Secure Data Processing.

Abstract: We present a system for efficient privacy-preserving multi-party querying (PPMQ) over federated graph databases. This framework offers a customisable and adaptable approach to privacy preservation using two different security protocols. The first protocol utilises standard secure multiparty computation (SMPC) protocols on the client side, enabling computations to be conducted on data without exposing the data itself. The second protocol is implemented on the server side using a combination of an SMPC protocol to prevent exposing the data to the clients and the use of encrypted hashing to prevent exposing the data to the server. We have conducted experiments to compare the efficiency of our PPMQ system with Neo4j Fabric, the off-the-shelf solution for querying federated graph databases, and with two previous systems, SMPQ and Conclave for secure multiparty querying. The results demonstrated that the execution times and overheads of PPMQ are comparable to those using Neo4j Fabric. Notably, our results reveal that the execution times and overheads of PPMQ outperform both SMPQ and Conclave, showcasing the better efficiency of our approach in preserving privacy within federated graph databases.

1 INTRODUCTION

Given the significance of data security, it is unsurprising that numerous techniques have been proposed and designed to enhance it. Secure multi-party computation (SMPC) stands out as a particularly intriguing example of a method developed for this purpose. According to (Cramer et al., 2015), SMPC is a cryptographic technique that enables a group of individuals to collaborate on computations while keeping their private data secret. It offers several advantages, including full data privacy, where no third parties can access the data regardless of their level of trustworthiness. Additionally, it eliminates the need to compromise between data usability and privacy, enabling data processing with high accuracy.

Nowadays, SMPC has a wide range of practical applications, including, but not limited to, detecting financial fraud (Sangers et al., 2019), aggregating model features from private datasets, and predicting heart disease(van Egmond et al., 2021). Moreover, SMPC can address trust-related concerns in various scenarios, such as secure elections (Alwen et al., 2015), auctions (Aly and Van Vyve, 2016), and secret sharing (Evans et al., 2018).

Until now, in the context of data processing, SMPC has primarily been utilised to safeguard relational databases (Volgushev et al., 2019; Poddar et al., 2020; Bater et al., 2016). More recently, applications of SMPC for other types of databases with different data models have been considered. This includes the system targeting graphs databases like GOOSE (Ciucanu and Lafourcade, 2020), which employs SMPC at the backend, while queries remain one-party only, and SMPQ (Al-Juaid et al., 2022) which implements multi-party queries. Compared to relational databases, graph databases provide a more flexible data model that is more efficient for some types of queries, such as traversal queries (Salehnia, 2017). The graph model employed by graph databases has proven advantageous in numerous scenarios, and these databases have had a wide variety of uses, including on social media platforms, such as Instagram, Twitter, and Facebook (Ciucanu and Lafourcade, 2020).

While SMPC applied to multi-party queries has shown promise, serious challenges remain, such as low performance. For example, our SMPQ system reported in (Al-Juaid et al., 2022) adopts the previously developed Conclave system (Volgushev et al.,

Aljuaid, N., Lisitsa, A. and Schewe, S.
Efficient and Secure Multiparty Querying over Federated Graph Databases.
DOI: 10.5220/0012757100003756

In *Proceedings of the 13th International Conference on Data Science, Technology and Applications (DATA 2024)*, pages 39-50
ISBN: 978-989-758-707-8; ISSN: 2184-285X

2019) for graph databases and optimizes processing, but still, significant overheads remain.

1.1 Our Contributions

We have developed a privacy-preserving multi-party query system (PPMQ) that leverages the Neo4j Fabric functionality extended with the Awesome Procedures on Cypher (APOC) library (Gu et al., pear), (Needham and Hodler, 2019).

In PPMQ, we altered and improved the architecture of our previous SMPQ model introduced in (Al-Juaid et al., 2022) by eliminating the Conclave layer, connecting directly to the JIFF server (Albab et al., 2019) to reduce the overheads. This system can handle a larger number of queries than SMPQ.

The system provides the user with two different security protocols. The first protocol involves using traditional SMPC protocols on the client side, enabling computations to be performed on the data without revealing it.

The second protocol, a novel approach implemented on the server side, utilises traditional SMPC protocols and involves hashing the data and masking it with a Diffie-Hellman (DH) key exchanged among all clients. This protects the data against access by the server. The choice between the two protocols is determined based on the computations the parties want to perform within the query itself.

We measured the execution times of both protocols to compare the performance of the PPMQ system with Neo4j Fabric (without encryption or SMPC) as well as with two existing SMPC-based systems: Conclave(Volgushev et al., 2019) and SMPQ (Al-Juaid et al., 2022).

The remainder of this paper is organised as follows: Section 2 presents the background to this work, followed by a review of related literature (Section 3). Next, our approach is outlined in Section 4. Following this, Section 5 presents the evaluation of the system and the experimental setup, followed by the results of our experiments. Finally, the paper concludes in Section 6, and we offer directions for future work.

2 BACKGROUND

2.1 Secure Multi-Party Computation (SMPC)

SMPC enables a group of parties to jointly compute a function while ensuring the confidentiality of their individual inputs. Imagine a scenario where several parties, denoted as P, each possess private data X that they want to use for a computation represented by function F. If these parties don't fully trust each other, they may choose to involve a trusted third party, denoted as Z, to perform the computation. In this setup, parties provide their data to Z, who computes F and returns the respective part of the result, Y, to each party.

However, SMPC offers an alternative approach. It enables parties to collaboratively compute F while preserving the privacy and security of their data. Regardless of the number of parties involved, each with their own private data, SMPC ensures that the computation of F is conducted confidentially. This guarantees that the outcome remains identical to what would be obtained if a trusted third party Z were to perform the computation.

We refer the reader to (Evans et al., 2018) for an overview of SMPC and related standard security definitions.

2.2 Neo4j and Cypher Query Language

In this work, Neo4j served as the implementation for graph databases (Guia et al., 2017). The graph data model employed in Neo4j comprises a collection of nodes and relationships. Nodes signify individual entities, with connections between them depicted using arrows (Miller, 2013). Each node in the graph may be assigned properties to describe the entity it represents. Moreover, labels can be attached to nodes, enhancing the ease of conducting efficient searches within the graph. Neo4j incorporates the Cypher query language for managing data within graph databases (Francis et al., 2018).

In our system implementation, we make use of Neo4j Fabric functionality (Gu et al., pear) along with the APOC library (Needham and Hodler, 2019). Neo4j Fabric operates on the principle of executing Cypher queries that simultaneously target multiple Neo4j graph databases, making it an ideal solution for federated databases.

In general, a federated database denotes a type of database management system that integrates multiple autonomous database systems into a single federated database. Similarly, a federated graph collaboratively integrates multiple sharded graphs, enabling the querying of all these graphs as a single big graph database (Azevedo et al., 2020). There are three main challenges in federated computation: privacy, efficiency, and effectiveness. Privacy involves maintaining the confidentiality of each owner's private data while facilitating collaborative computing. Efficiency is influenced by additional communication and com-

putation. Effectiveness pertains to conducting accurate analysis in the presence of multiple distributed datasets (Tong et al., 2023).

The APOC library contains more than 450 procedures and functions designed to assist in various common tasks, including data integration, cleaning, conversion, and general-purpose helper functions. Figure 1 illustrates an example general Cypher query involving two parties with two databases extended using Neo4j Fabric and the intersection procedure from the APOC library.

```
CALL { USE graph1
    MATCH (node1: label1)-[:Relationship]→(node2:label2)
    RETURN node2 AS Output1 }
 CALL {USE graph2
    MATCH (node1: label1)-[:Relationship]→(node2:label2)
    RETURN node2 AS Output2}
RETURN apoc.coll.intersection (Output1, Output2) AS output;
```

Figure 1: Example of a general Cypher query utilising two parties with two databases.

3 RELATED WORK

3.1 Implementation of SMPC

Until recently, SMPC was primarily a subject of theoretical exploration. However, in recent times, there has been a notable effort to apply it in real-world applications (Evans et al., 2018). Examples of SMPC implementations used in various contexts include JIFF (Albab et al., 2019), Oblivm (Liu et al., 2015), GraphSC (Nayak et al., 2015), and Sharemind (Bogdanov et al., 2008).

JIFF (Albab et al., 2019) is a JavaScript library designed for scenarios involving distributed data among multiple entities, facilitating SMPC. It employs a server to manage encrypted messages exchanged between participants, operating under a semi-honest security model. JIFF utilises Shamir's secret sharing method (Shamir, 1979) for secure computation. In SMPC, the private information of each participant is divided into shares, and distributed among multiple parties. These shares individually hold no meaning but can be collectively combined to reconstruct the original secret.

When JIFF operates as a server, each party acts as a client. Throughout the computation process, input data from participants is divided into shares and encrypted using the public keys of the parties. Each party possesses shares from others, including their own. The computation is performed on these shares, and the output is displayed to all participants without disclosing the underlying data involved in the calculation. The architecture of JIFF is illustrated in Figure 2. In the Shamir secret sharing scheme (Shamir, 1979), two operations are employed: share generation and reconstruction. During share generation, shares are created from a given secret input, where the secret is divided into several shares and distributed among n parties. The reconstruction process defines the minimum number of parties required to reconstruct the original secret input.

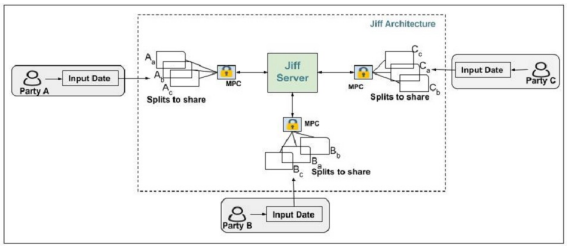

Figure 2: JIFF architecture and components.

ObliVM (Liu et al., 2015) is a secure computation framework using ObliVM-lang, akin to Java. It employs a two-party garbled circuit protocol, supports arbitrary-sized integers, fixed-size integers, and an efficient Oblivious RAM (ORAM) scheme. The authors demonstrate efficient complex arithmetic, like Karatsuba multiplication, in ObliVM-lang.

GraphSC is a secure computation framework leveraging ObliVM for graph parallelism. It relies on ObliVM (Liu et al., 2015) and enables secure two-party computation where one party garbles and the other evaluates data. Performance evaluations show GraphSC processes data roughly ten times faster than ObliVM.

Sharemind is a high-speed, secure computation framework using a three-party hybrid protocol and additive secret-sharing scheme. Its roles include clients, servers, and outputs. Sharemind operates within a finite ring, processing data securely while ensuring privacy and scalability.

In this work, we will utilise JIFF (Albab et al., 2019) as the backend for our PPMQ system to implement SMPC protocols.

3.2 SMPC for Data Processing

There have been recent efforts to integrate SMPC with databases to enhance data security. For instance, (Volgushev et al., 2019) proposes Conclave, a query compiler used with relational databases. This compiler converts queries into a combination of data-parallel, local cleartext processing, and small SMPC steps. The system rewrites queries to reduce the cost of SMPC processing and improve scalability. They recommend forwarding the modified query to JIFF (Albab et al., 2019), which serves as the backend SMPC system.

In (Poddar et al., 2020), the Senate system is introduced, enabling collaborative execution of analytical SQL queries while preserving data confidentiality. Unlike previous systems, the Senate offers protection against malicious parties rather than just semi-honest modes.

Additionally, in (Liagouris et al., 2021), the authors proposed Secrecy, a relational SMPC framework based on replicated secret sharing. It partitions data into three shares where each party handles two shares to execute query segments securely.

Furthermore, (Bater et al., 2016) presents SMCQL, a system converting SQL queries into secure multiparty computations. Users submit queries to an honest broker, acting as a trusted third party, responsible for processing queries securely and delivering results.

In another study by (Bater et al., 2020), they used SMCQL to develop the Secure Approximate Query Evaluator (SAQE) system, enhancing SQL query security. SAQE follows a two-stage approach: planning and execution. The client optimizes and executes query plans, while query execution occurs on the server among data owners, using SMPC protocols. They collaborate to execute queries across databases and share results with the client.

Bater et al. expanded on the SMCQL system and developed a system called Shrinkwrap (Bater et al., 2018), using two-party secure computations. While improving SMCQL's efficiency, it's important to note that some information is disclosed in the process.

In (Wang and Yi, 2021), the authors introduce Secure Yannakakis, a modified version of the classical Yannakakis algorithm tailored for secure two-party computation. This protocol enables parties to assess free-connex join-aggregate queries while safeguarding data confidentiality. Results demonstrate a notable efficiency enhancement compared to the current method relying on Yao's garbled circuit.

VaultDB, outlined in (Rogers et al., 2022), is a framework for secure SQL query computation over private data from diverse sources. It utilises the EMP toolkit as its SMPC backend. The authors evaluated it with a dataset from a Clinical Research Network, covering data from almost 13 million patients. Their results highlight the efficiency and scalability of the system in distributed clinical research analyses while maintaining patient record privacy.

Scape (Han et al., 2022) is a scalable collaborative analytics system for private databases with malicious security. It enables secure sharing among three non-colluding computing parties, allowing users to execute various SQL queries. Benchmark results show Scape is up to 25 times faster than the Secrecy framework (Liagouris et al., 2021).

Sequre, outlined in (Smajlović et al., 2023), is a Python-based framework for SMPC in bioinformatics. It prioritises high performance and incorporates automatic compile-time optimizations to boost efficiency and speed. Through secret-sharing, Sequre enables secure computation of biomedical data while ensuring faster query execution and analysis tasks.

Hu-Fu (Tong et al., 2022) is a system designed for secure and efficient spatial queries within a data federation. It minimizes secure operations and enhances query processing performance compared to existing solutions while ensuring robust security across various spatial databases.

Contrastingly, in (He et al., 2015) and (Ciucanu and Lafourcade, 2020), the authors explore SMPC's application for single-party querying. They introduce the SDB system, a cloud-based relational database involving two parties: the data owner (DO) and the server provider (SP). Sensitive data is split into two shares: one held by the DO (item key) and the other by the SP (ciphertext). SMPC (secret sharing) is used between the DO and the SP. When a user submits an SQL query, the SDB proxy in the DO segment transforms sensitive column queries into corresponding User-Defined Functions (UDFs) at the SP. These altered queries are then sent to the SP, and the encrypted results are returned to the SDB proxy for decryption before being provided to the user.

The GOOSE framework, detailed in (Ciucanu and Lafourcade, 2020), shares similarities with SDB and primarily focuses on securing outsourced data within a Resource Description Framework (RDF) graph database using SMPC secret sharing. Graph data is divided into three components, encrypted, and distributed across the cloud. All components are multi-party, preventing any single party from accessing the complete graph, query, or outcomes. Data transmissions between parties are encrypted using the AES algorithm.

Initially, multi-party query security in graph databases was outlined in the SMPQ (Al-Juaid et al., 2022) framework. Subsequent advancements transformed SMPQ into a fully automatic solution, expanding its query-handling capabilities with significantly improved performance (Al-Juaid et al., 2023). Notably, this system is built upon the Conclave system, utilizing the SMPC protocol to secure relational databases.

We introduce PPMQ, a system designed to secure multiparty queries on graph databases. The PPMQ model is inspired by SMPQ, in which we altered the architecture by eliminating the Conclave layer and connecting directly to the JIFF server(Albab et al.,

2019) to reduce the overheads. We implemented PPMQ using JavaScript and built it on top of the JIFF library, serving as an implementation of SMPC protocols.

Table 1 provides a comparative analysis between our proposed PPMQ system and all mentioned above systems.

4 DESIGN OF THE PPMQ FRAMEWORK

In this section, we first introduce the entities involved in the PPMQ framework and then provide an overview of how it works.

4.1 Involved Entities

The architecture of our PPMQ system is illustrated in Figure 3. The system involves the following entities:

1. **Data Owners.** It enables multiple data owners, denoted by $P_1, P_2, \ldots, P_n$, to jointly execute a single query on the union of their own separate and private databases, once they reach an agreement.
2. **Neo4j Databases.** The different data owners make their respective Neo4j databases available for the system, to execute the sub-query for each party using their Neo4j database.
3. **JIFF Server.** This entity represents a server that provides SMPC protocols like secret sharing to be used in joint querying.

4.2 Security Guarantees and Assumption

PPMQ is designed on top of the JIFF library, which follows a semi-honest security model. Parties agree via out-of-band mechanisms on the query to run, and all parties faithfully execute the protocol. In the semi-honest security model, parties adhere to the protocol but may attempt to learn private information by analysing their messages or colluding with other parties. The security of PPMQ is built upon the robustness of the underlying cryptographic primitives employed by JIFF, encompassing secure encryption algorithms, key exchange protocols, and cryptographic hash functions.

4.3 Query Agreement

The PPMQ system allows for joint querying by two or more parties. After determining the number of parties involved in performing the query using the SMPC protocol, they should agree on a computation ID. In our current system, this computation ID (ComID) can be considered an agreement to apply the query using their respective databases (Albab et al., 2019). In a further enhancement of the system, we will use the threshold shared signatures method (Tang et al., 2023).

4.4 PPMQ Overview

The PPMQ system works by providing users with the ability to execute queries using two different security protocols. One option is to use traditional SMPC protocols on the client side, allowing computations to be performed on the data without exposing it. The second protocol is implemented on the server side using SMPC protocols and involves hashing the data to use the hashed value. This protects the data against access by the server, assuming that hashing is not reversible, contributing to the security of the data. The choice between the two protocols is determined based on the computations the parties want to perform within the query itself. If the query involves performing arithmetic operations, including secure addition, multiplication, division, comparison, or sorting, it will be done on the client side. However, if the purpose of the computation in the query is to find the intersection among the parties' private data, this computation will be carried out using the second protocol on the server side. The types of queries covered include those involving arithmetic operations such as secure addition, multiplication, division, comparison, or sorting. Additionally, queries aimed at finding the intersection among the parties' private data are handled using the second protocol on the server side. Furthermore, as part of our ongoing development efforts, we intend to extend the coverage to include graph traversal queries over multiple parties databases.

4.4.1 Protocol 1: Client-Based

To apply traditional SMPC protocols on the client side, leveraging the functionality provided by JIFF, the sub-query results obtained from each party are transmitted to the JIFF server. At the JIFF server, the SMPC protocol (Secret Sharing) is employed to divide the data into shares. Subsequently, JIFF utilises a server to store and route encrypted messages exchanged among the participating parties for computation purposes. JIFF's functionality encompasses performing arithmetic operations, including secure addition, multiplication, and division of two or more secret-shared numbers. This capability enables parties to collaboratively compute a joint function without disclosing their respective inputs. Furthermore,

Table 1: SMPC for data processing.

Framework	*Parties supported*	*SMPC*	*Framework backend*	*Trust Party*	*No.Data owners*	*Data Model*	*Query language/API*	*Available implementation*	*Development language*
Conclave (Volgushev et al., 2019)	>= 2	Secret Sharing	JIFF	Yes	>= 2	Relational DB	SQL/LINQ	Yes	Python
Congregation	>= 2	Secret Sharing	JIFF	No	>= 2	Relational DB	SQL	Yes	Python
SMCQL (Bater et al., 2016)	2	Garbled Circuits/ ORAM	ObliVM	No	2	Relational DB	SQL	Yes	Java
Senate (Poddar et al., 2020)	2	Garbled Circuits	N/A	No	2	Relational DB	SQL	No	-
SAQE (Bater et al., 2020)	2	Garbled Circuits	ObliVM	No	2	Relational DB	SQL	No	-
Shrinkwarp (Bater et al., 2018)	2	Garbled Circuits/ ORAM	ObliVM	No	2	Relational DB	SQL	No	-
Secrecy (Liagouris et al., 2021)	3	Repl.Secret Sharing	N/A	No	3	Relational DB	SQL	No	C
SecureYannakakis (Wang and Yi, 2021)	2	Garbled Circuits	N/A	No	2	Relational DB	SQL	Yes	C++
VaultDB (Rogers et al., 2022)	2	Garbled Circuits	EMP-Toolkit	No	2	Relational DB	SQL	No	C++
Scape (Han et al., 2022)	3	Repl.Secret Sharing	Frigate	No	3	Relational DB	SQL	No	-
Sequre (Smajlović et al., 2023)	3	Secret Sharing	N/A	Yes	3	Relational DB	SQL	Yes	Python/C++
Hu-Fu (Tong et al., 2022)	>= 2	N/A	N/A	Yes	n	Relational DB	SQL	Yes	Java
SMPQ (Al-Juaid et al., 2022)	>= 2	Secret Sharing	Conclave/JIFF	No	>= 2	GraphDB	Cypher	No	Python
SDB (Wong et al., 2014; He et al., 2015) [1]	N/A	Secret Sharing	N/A	No	1	Relational DB	SQL	No	-
GOOSE (Ciucanu and Lafourcade, 2020) [2]	N/A	Secret Sharing	N/A	No	1	GraphDB	SPARQL	Yes	Python
PPMQ	>= 2	Secret Sharing	JIFF	No	>= 2	GraphDB	Cypher	No	JavaScript

*Both [1] and [2] use SMPC as backend over a database; they do not support multi-party user queries

JIFF supports comparison operations, allowing parties to determine the relative magnitude of their inputs, as well as sorting operations, permitting parties to establish the order of their inputs without revealing the actual values. Additionally, we have developed a custom function for identifying the intersection between two or more secret-shared strings, further extending the functionality of JIFF. Algorithm 1 illustrates the phases involved in client-based query protocol.

To illustrate how this works, consider a scenario with three parties aiming to execute one of the queries from the subsection 5.2 specifically, Q2, which is a query to **count the number of students who scored 7 and are common across all databases**. The syntax for the joint query will be as follows:

```
    CALL {USE db1
    MATCH (n:Prof) -[:Guide]-> (m:Student)
    where m.Score= 7
    RETURN count(m) as db1}
    CALL {USE db2
    MATCH (n:Prof) -[:Guide]-> (m:Student)
    where m.Score= 7
    RETURN count(m) as db2}
    CALL {USE db3
    MATCH (n:Prof) -[:Guide]-> (m:Student)
    where m.Score= 7
    RETURN count(m) as db3}
RETURN apoc.coll.sum (db1,db2,db3 ) AS cnt
```

In this example, each party executes a sub-query to count students with a score of 7 in its respective database. The results ($db1$, $db2$, $db3$) are transmitted to the server as shares. These shares are then distributed among the parties to compute intermediate results. Subsequently, based on these intermediate results, the final result of the joint query Q is determined.

Algorithm 1: Client-Based Query Execution.

Require: Q: Joint query
Require: P: Set of parties $P_1, P_2, \ldots, P_n$ for sharing computations.
Require: *ComID*: Shared computation ID as proof of agreement.
Require: P_{Id}: Unique ID for each P when connecting to the server.
Require: $R(Q_i)$: Result of subquery Q_i

Connect to the System
1. Decide the number of parties (P_i) involved in the computation.
2. Use shared *ComID* to agree on the computation.

Preprocessing Phase
1. Each P_i submits Q to the system.
2. Parse Q into Q_i based on P_{Id} for each P_i.
3. Execute Q_i on each P_i's Neo4j database.

Computation Phase
1. Each P_i sends R(Q_i) to the server as shares.
2. Distribute shares among P_i, where each P_i receives one share from others.
3. Each P_i computes the intermediate result using a subset of these shares.

Reconstruction Phase
1. Each P_i sends the intermediate result to P_{i+1}.
2. Given these results, find the final result of Q.

4.4.2 Protocol 2: Server-Based

Due to the potential lack of information when performing the intersection on the client side, each party receives shares from the other parties (without knowing the source) and identifies duplicate names as the intersection. To solve this issue, we propose an alternative protocol that performs the intersection on the server side while incorporating an additional method to enhance the security of the information against po-

tential server access. In our current version of PPMQ, this protocol supports performing only the intersection operation. As an enhancement to the system, we intend to use this protocol to study how to perform *traversal queries*, paradigmatic for graph databases, between multiple parties while preserving the privacy of their data. Algorithm 2 outlines the phases of query execution on the server side.

Algorithm 2: Server-Based Query Execution.

Require: Q: Joint query
Require: P: Set of parties $P_1, P_2, \ldots, P_n$ for sharing computations.
Require: *ComID*: Shared computation ID as proof of agreement.
Require: P_{Id}: Unique ID for each P when connecting to the server.
Require: *Dh*: Key generated and exchanged using the DH algorithm.
Require: $R(Q_i)$: Result of subquery Q_i

Connect to the System
1. Decide how many parties (P_i) are involved in the computation.
2. Use shared *ComID* to agree on the computation.
3. After (P_i) connects to the system, generate *Dh* and exchange it between P_n using the JIFF server.

Preprocessing Phase
1. Each P_i submits Q to the system.
2. Parse Q into Q_i based on P_{Id} for each P_i.
3. Execute Q_i on each P_i's Neo4j database.
4. Each P_i hashs $R(Q_i)$ then, XORed using exchanged *Dh*.

Computation Phase
1. Each (P_i) sends the newly hashed $R(Q_i)$ to the server as shares.
2. The server computes the intersection on the hashed results received from (P_i).
3. The result of the computation is distributed to each (P_i).

Reconstruction Phase
1. Each P_i performs XOR on the result using *Dh*.
2. The final result of Q is determined from these XORed results.

To illustrate how this works, let's consider an example scenario involving two parties. They want to run a query to find **the names of people who appear in both of their databases and were born in 1977**. The syntax for the joint query will be as follows:

```
CALL{
USE db1
MATCH (m:Person) WHERE m.born= 1977
RETURN DISTINCT(m.name)  AS out1
}
CALL{
USE db2
MATCH (m:Person)
WHERE m.born >= 1977 and m.born<=1980
RETURN DISTINCT(mm.name)  AS out2
}
RETURN apoc.coll.intersection
(out1,out2)
```

In this example, each party executes a sub-query to identify distinct names based on certain conditions. The results (out_1, out_2) are sent as shares to a server, which computes the intersection of these results without revealing underlying data. The final result of the joint query Q is determined, representing the shared names between them, which is returned to the participating parties for privacy-preserving data collaboration.

4.5 Query Workflow

Figure 3 outlines the query flow in the PPMQ system. Multiple data owners (P_1 to P_n) collaboratively execute a joint query on separate graph databases. In step (1), they agree on a joint query and establish a shared computation ID. Moving to step (2), parties submit the query, which is then translated into sub-queries depending on parties IDs. In step (3), each party executes their sub-query on their Neo4j database. The sub-query could be the same across all parties, or it can be an arbitrary sub-query. From step(1) to step (3), the same steps will be on both the client and server sides.

4.5.1 Query Workflow: Client Side

In step (4), sub-query results are sent to the JIFF server, utilising SMPC protocols. Privacy is ensured as results are passed as shares utilise Shamir's secret sharing method (Shamir, 1979). In step (5), shares are distributed among parties to perform a secure operation and compute the final query results. Finally, in step (6), the exclusive outcome is revealed to the parties who initiated the query, maintaining the confidentiality of the underlying data.

4.5.2 Query Workflow: Server Side

In Step (4), a random key (Dh) is generated using DH key exchange between the parties (Maurer and Wolf, 1998). Then, in Step (5), when the query results are obtained and before being passed to the JIFF server, each party hashes their result using SHA-256. This is followed by XORing the hashed value using Dh. Moving on to Step (6), within the JIFF server, SMPC divides each party's private data into smaller shares

but does not distribute them among various parties. Instead, the server performs the computation to find the intersection. Finally, in Step (7), the final result is revealed to the parties who initiated the query.

5 SYSTEM EVALUATION

We evaluate our proposed system, PPMQ, and investigate its efficiency. The goal of the experiments is to answer the following questions: We evaluate the efficiency of our proposed system, PPMQ, addressing the following questions:

- **RQ1.** How effective is our system in ensuring secure multiparty queries, and what is its efficiency in terms of performance?
- **RQ2.** Does the data size impact the system's efficiency?
- **RQ3.** How does PPMQ's performance compare to existing systems like Neo4j Fabric, Conclave, and SMPQ?

5.1 Data Sets

To validate our proposed PPMQ system and assess its efficiency, we executed 15 queries using four distinct datasets. These datasets were sourced from three different parties, each using separate Neo4j databases.

The first dataset, created by us, contains data from professors and students, featuring 58 nodes and 29 edges. This dataset serves as a small-scale example for testing purposes. The second dataset is derived from an open Neo4j database example called the 'Movie' dataset (Neo4j, 2007). We modified the nodes by adding and removing some, establishing varying party sizes. The graph within this dataset consists of 563 nodes with 785 edges, enabling a more comprehensive evaluation. The third dataset, named the 'POLE' dataset, is a large-scale dataset providing open crime data for Manchester, UK, spanning August 2017 (Hunger, 2020). It encompasses 61,521 nodes with 105,840 relationships. Finally, we developed the 'Car Location' dataset exclusively using numerical data. Across all the databases in this dataset, a total of 100 nodes are interconnected by 63 relationships. This dataset was specifically designed to facilitate a comparison between our system and the Conclave system, given that the latter is designed for handling solely numerical data.

For clarification, we used the first dataset to execute queries Q1 to Q4, while the Movie dataset was used for queries Q5 to Q8. Additionally, queries Q9 to Q11 were executed using the POLE dataset, while the final set of queries, Q12 to Q15, was conducted using the last dataset. Table 2 provides further details about each dataset, including node and relationship counts for each database owned by distinct data owners.

Table 2: Details regarding the four datasets employed for conducting the experiments.

Data sets	Database	No. of Nodes	No. of Relationships
Prof-student	DB1	32	16
	DB2	10	5
	DB3	16	8
Movie	DB1	203	269
	DB2	172	254
	DB3	188	262
POLE	DB1	61521	105840
	DB2	61521	105840
	DB3	61521	105840
Car-Location	DB1	44	31
	DB2	24	15
	DB3	32	17

5.2 Queries

Below is the list of the 15 queries used to validate our system[1].

- **Q1.** Count how many students there are in common between all DBs.
- **Q2.** Count the number of students who scored 7 and are common across all databases.
- **Q3**: Find the names of the common students across all the databases with scores of 9 or above.
- **Q4.** Find names of the common students across all databases with scores of 7.
- **Q5.** Count the number of movies with the actor Tom Hanks that are common across all databases.
- **Q6.** Find the names of the movies that are common across all databases with the actor Tom Hanks.
- **Q7.** Find the names of all actors who were born in 1974.
- **Q8.** Finds the sum of all nodes in the movie DB for all parties.
- **Q9.** Find the total number of robbery crimes across all databases for the year 2017 in Manchester.
- **Q10.** Determine the location with the highest frequency of recorded burglaries in common across all databases.
- **Q11.** Find which crime Inspector Morse investigated is listed in all databases.

[1] all Cypher queries can be found in the Appendix available at https://doi.org/10.5281/zenodo.11048030

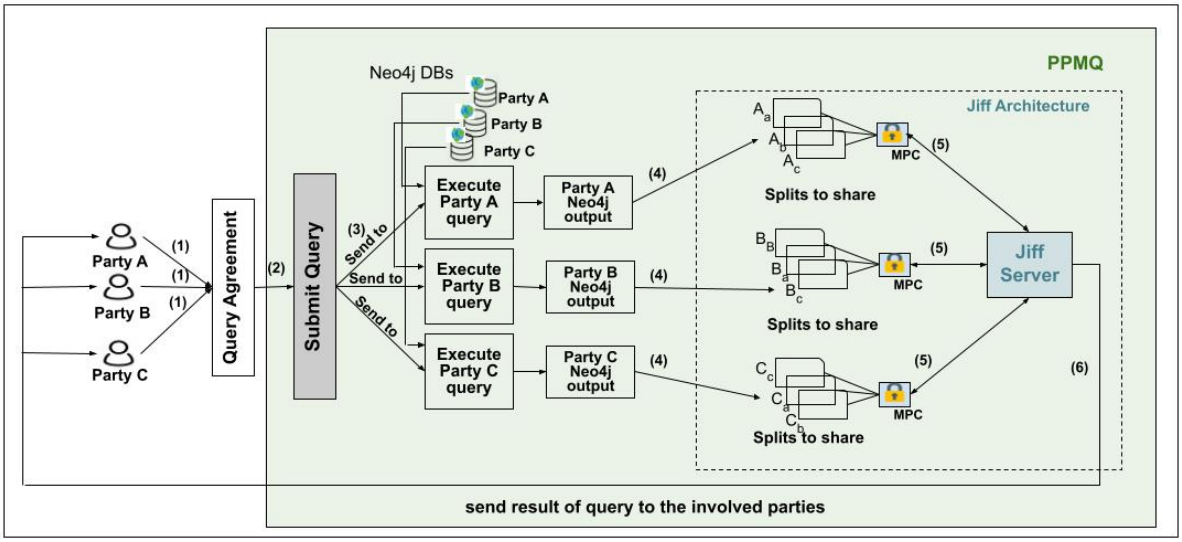

Figure 3: PPMQ architecture and components.

- **Q12.** Find the number of cars present at each specific location.
- **Q13.** Find the total count of cars with ID=1 across all the databases.
- **Q14.** Combine two databases by utilising the node id of the professors whose students have a grade equal to or higher than 9.0 in either of the databases.
- **Q15.** Retrieve information from two databases by selecting the scores of all students enrolled in the Math course from both databases.

The experiments were conducted on a desktop PC running Windows 11 with an Intel Core i7 processor clocked at 1.5 GHz and 16.00 GB of RAM, utilizing a local database for each party. Execution times, representing the duration for all parties to obtain query results, were measured using the *performance.now()* function in JavaScript, providing high-resolution timestamps in milliseconds. We executed 10 iterations for each query. Table 3 presents mean running times and standard deviation for all queries executed on PPMQ and on Neo4j Fabric, the popular framework for federated graph databases which does not use any SMPC protocols. The results show comparable execution times and overheads for PPMQ and Neo4j Fabric (around double). Notably, even when executing Q9-Q11 with a large dataset, the performance remains acceptable, ranging from 79 to 208 ms. The evaluation covered a range of queries to assess the performance of our system across different query types, addressing RQ1. Furthermore, we explored the impact of data size on system efficiency using the large-scale 'POLE' dataset, addressing RQ2.

Table 3: Execution times for Q1– Q15 when using PPMQ and Neo4j Fabric.

	PPMQ system		Neo4j Fabric	
Query	**Mean (ms)**	**Standard Deviation**	**Mean(ms)**	**Standard Deviation**
Q1	79.4	19.2	51.5	19.3
Q2	99.7	59.3	36.5	11.3
Q3	90.26	49.5	65.4	15.63
Q4	76.5	17.4	35.3	15.8
Q5	67.3	16.1	24.2	13.5
Q6	70.86	17.6	44.6	14.8
Q7	67.3	17.1	35	10.5
Q8	72.5	23.9	14.5	3.74
Q9	79.5	29.7	18	9.8
Q10	208.5	58.6	104.7	40.6
Q11	72.5	27.5	71.3	20.5
Q12	57.7	13.3	46	7.8
Q13	51.6	14.4	54.1	13.5
Q14	83.6	22.4	59.2	13.1
Q15	77.6	14.1	32.4	12.2

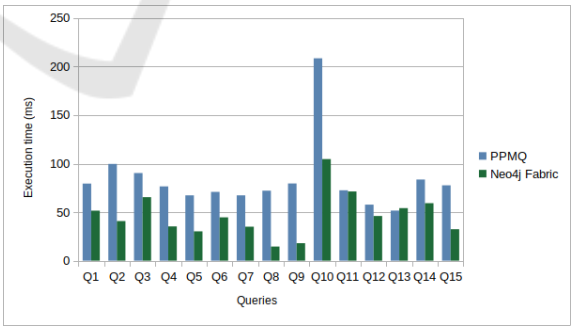

Figure 4: Execution times for Q1– Q15 when using PPMQ and Neo4j Fabric.

5.3 Comparison with Prior Work

In this subsection, we aim to compare our system, PPMQ, with two existing systems that leverage SMPC for data processing. The first system is Conclave (Volgushev et al., 2019), which uses SMPC to secure the relational database and serves as the back-

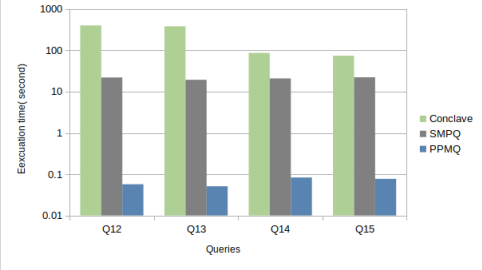

Figure 5: Execution times for Q9– Q12 when using Conclave, SMPQ, and PPMQ.

end for the second system, SMPQ (Al-Juaid et al., 2022) which focuses on secure multiparty queries in graph databases. To ensure a fair comparison, we executed the same queries used in our experiments using their SQL equivalents.

Subsequently, we compared PPMQ with SMPQ (Al-Juaid et al., 2022), which is built atop the Conclave system for secure multiparty queries on a graph database. However, when attempting to compare our system with Conclave, we encountered a limitation: Conclave only supports numerical data. Consequently, executing queries Q1-Q11 was not possible. To address this restriction, we utilized the 'Car location' dataset, which exclusively consists of numerical data, to perform queries Q12-Q15.

Running these queries in the Conclave system resulted in execution times ranging from approximately 402 to 74 seconds. Subsequently, executing the same queries in the SMPQ system reduced the time to 22 to 19 seconds. This performance improvement was achieved by eliminating the sorting function used in the Conclave system after obtaining the query result.

Notably, PPMQ demonstrated a significant advancement in query execution time compared to both Conclave and SMPQ. The removal of the Conclave layer allowed PPMQ to streamline the query execution process, eliminating unnecessary computational steps and reducing overall latency. As a result, PPMQ achieved execution times of less than a second for the same set of queries, showcasing its efficiency and effectiveness in secure multiparty query processing. Table 4 and Figure 5 illustrate the comparison of execution times when running the mentioned queries using Conclave, SMPQ, and PPMQ, thus providing insights into addressing research question RQ3.

Table 4: Comparison of execution times.

Query	**Conclave**(sec)	**SMPQ**(sec)	**PPMQ**(sec)
Q12	402.3	22.05	0.057
Q13	382.3	19.36	0.051
Q14	86.88	20.89	0.083
Q15	74.1	22.28	0.077

6 CONCLUSION

We have developed a framework for securing multiparty queries over federated graph databases based on SMPC protocols. Our system has been implemented using the JIFF server as a backend for SMPC protocols. Furthermore, we have expanded the system to automatically parse queries based on the party ID, enabling the execution of the correct sub-query. However, certain Cypher query language features, such as correlated queries, were not tested in the current system, as they were beyond the scope of this paper. In future work, we plan to extend the system to handle traversal queries between multiple private databases using SMPC protocols.

REFERENCES

Al-Juaid, N., Lisitsa, A., and Schewe, S. (2022). Smpg: Secure multi party computation on graph databases. In *ICISSP*, pages 463–471.

Al-Juaid, N., Lisitsa, A., and Schewe, S. (2023). Secure joint querying over federated graph databases utilising smpc protocols. In *Proceedings of the 9th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pages 210–217. INSTICC, SciTePress.

Albab, K. D., Issa, R., Lapets, A., Flockhart, P., Qin, L., and Globus-Harris, I. (2019). Tutorial: Deploying secure multi-party computation on the web using JIFF. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 3–3. IEEE.

Alwen, J., Ostrovsky, R., Zhou, H., and Zikas, V. (2015). Incoercible multi-party computation and universally composable receipt-free voting. In *Advances in Cryptology–CRYPTO 2015: 35th Annual Cryptology Conference,*, volume 9216, pages 763–780. Springer Berlin Heidelberg.

Aly, A. and Van Vyve, M. (2016). Practically efficient secure single-commodity multi-market auctions. In *International Conference on Financial Cryptography and Data Security*, pages 110–129. Springer.

Azevedo, L. G., de Souza Soares, E. F., Souza, R., and Moreno, M. F. (2020). Modern federated database systems: An overview. *ICEIS (1)*, pages 276–283.

Bater, J., Elliott, G., Eggen, C., Goel, S., Kho, A., and Rogers, J. (2016). SMCQL: secure querying for federated databases. *arXiv preprint arXiv:1606.06808*.

Bater, J., He, X., Ehrich, W., Machanavajjhala, A., and Rogers, J. (2018). Shrinkwrap: efficient sql query processing in differentially private data federations. *Proceedings of the VLDB Endowment*, 12(3):307–320.

Bater, J., Park, Y., He, X., Wang, X., and Rogers, J. (2020). SAQE: practical privacy-preserving approximate query processing for data federations. *Proceedings of the VLDB Endowment*, 13(12):2691–2705.

Bogdanov, D., Laur, S., and Willemson, J. (2008). Sharemind: A framework for fast privacy-preserving computations. In *Computer Security-ESORICS 2008: 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings 13*, pages 192–206. Springer.

Ciucanu, R. and Lafourcade, P. (2020). GOOSE: A secure framework for graph outsourcing and sparql evaluation. In *34th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec'20). Accepté, à paraître*.

Cramer, R., Damgård, I. B., and Nielsen, J. B. (2015). *Secure multiparty computation*. Cambridge University Press.

Evans, D., Kolesnikov, V., and Rosulek, M. (2018). A pragmatic introduction to secure multi-party computation. *Found. Trends Priv. Secur.*, 2:70–246.

Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., and Taylor, A. (2018). Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1433–1445.

Gu, Z., Corcoglioniti, F., Lanti, D., Mosca, A., Xiao, G., Xiong, J., and Calvanese, D. (to appear). A systematic overview of data federation systems. *Semantic Web*.

Guia, J., Soares, V. G., and Bernardino, J. (2017). Graph databases: Neo4j analysis. In *ICEIS (1)*, pages 351–356.

Han, F., Zhang, L., Feng, H., Liu, W., and Li, X. (2022). Scape: Scalable collaborative analytics system on private database with malicious security. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1740–1753. IEEE.

He, Z., Wong, W. K., Kao, B., Cheung, D., Li, R., Yiu, S., and Lo, E. (2015). SDB: A secure query processing system with data interoperability. *Proc. VLDB Endow.*, 8:1876–1879.

Hunger, M. (2020). neo4j-graph-examples/pole. https://github.com/neo4j-graph-examples/pole/. Accessed: 2023-12-28.

Liagouris, J., Kalavri, V., Faisal, M., and Varia, M. (2021). Secrecy: Secure collaborative analytics on secret-shared data. *arXiv preprint arXiv:2102.01048*.

Liu, C., Wang, X. S., Nayak, K., Huang, Y., and Shi, E. (2015). ObliVM: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy*, pages 359–376.

Maurer, U. and Wolf, S. (1998). Diffie-hellman, decision diffie-hellman, and discrete logarithms. In *Proceedings. 1998 IEEE International Symposium on Information Theory (Cat. No. 98CH36252)*, page 327. IEEE.

Miller, J. J. (2013). Graph database applications and concepts with Neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, volume 2324.

Nayak, K., Wang, X. S., Ioannidis, S., Weinsberg, U., Taft, N., and Shi, E. (2015). Graphsc: Parallel secure computation made easy. In *2015 IEEE Symposium on Security and Privacy*, pages 377–394. IEEE.

Needham, M. and Hodler, A. E. (2019). *Graph algorithms: practical examples in Apache Spark and Neo4j*. O'Reilly Media.

Neo4j (2007). built-in examples: Movie-graph. https://neo4j.com/developer/example-data/#built-in-examples/. Accessed: 2023-12-28.

Poddar, R., Kalra, S., Yanai, A., Deng, R., Popa, R. A., and Hellerstein, J. M. (2020). Senate: A maliciously-secure mpc platform for collaborative analytics. *arXiv e-prints*, pages arXiv–2010.

Rogers, J., Adetoro, E., Bater, J., Canter, T., Fu, D., Hamilton, A., Hassan, A., Martinez, A., Michalski, E., Mitrovic, V., et al. (2022). Vaultdb: A real-world pilot of secure multi-party computation within a clinical research network. *arXiv preprint arXiv:2203.00146*.

Salehnia, A. (2017). Comparisons of relational databases with big data: a teaching approach. *South Dakota State University Brookings, SD 57007*, pages 1–8.

Sangers, A., van Heesch, M., Attema, T., Veugen, T., Wiggerman, M., Veldsink, J., Bloemen, O., and Worm, D. (2019). Secure multiparty pagerank algorithm for collaborative fraud detection. In *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*, pages 605–623. Springer.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Smajlović, H., Shajii, A., Berger, B., Cho, H., and Numanagić, I. (2023). Sequre: a high-performance framework for secure multiparty computation enables biomedical data sharing. *Genome Biology*, 24(1):1–18.

Tang, G., Pang, B., Chen, L., and Zhang, Z. (2023). Efficient lattice-based threshold signatures with functional interchangeability. *IEEE Transactions on Information Forensics and Security*, 18:4173–4187.

Tong, Y., Pan, X., Zeng, Y., Shi, Y., Xue, C., Zhou, Z., Zhang, X., Chen, L., Xu, Y., Xu, K., et al. (2022). Hu-fu: Efficient and secure spatial queries over data federation. *Proceedings of the VLDB Endowment*, 15(6):1159.

Tong, Y., Zeng, Y., Zhou, Z., Liu, B., Shi, Y., Li, S., Xu, K., and Lv, W. (2023). Federated computing: Query, learning, and beyond. *IEEE Data Eng. Bull.*, 46(1):9–26.

van Egmond, M. B., Spini, G., van der Galien, O., IJpma, A., Veugen, T., Kraaij, W., Sangers, A., Rooijakkers, T., Langenkamp, P., Kamphorst, B., et al. (2021). Privacy-preserving dataset combination and lasso regression for healthcare predictions. *BMC medical informatics and decision making*, 21(1):1–16.

Volgushev, N., Schwarzkopf, M., Getchell, B., Varia, M., Lapets, A., and Bestavros, A. (2019). Conclave: secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–18.

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

Wong, W. K., Kao, B., Cheung, D. W. L., Li, R., and Yiu, S. M. (2014). Secure query processing with data interoperability in a cloud database environment. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1395–1406.