Secure Joint Querying Over Federated Graph Databases Utilising SMPC Protocols

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

Keywords: Graph Databases, Secure Multi-Party Computation, Federated Databases, Secure Data Processing.

Abstract: We present a methodology for secure joint querying over federated graph databases based on secure multiparty computation (SMPC). Using SMPC instead of (or in addition to) encryption lifts reliance on the security of the encryption mechanism. The secret keeping is, instead, guaranteed by an SMPC protocol that protects the information required to answer a given query so that it is not shared in full on any communication line. We have recently outlined how this could be done in principle in a position paper, albeit with a sluggish implementation with an enormous computational overhead that rendered it unusable in practice. In this paper, we proposed an approach by better integrating it with the SMPC protocol, implementing it in JIFF, and covering the joint functionalities and languages of Conclave, Neo4j Fabric, and APOC. When implementing our prototype, we demonstrate how small queries can be served in fractions of a second, thus improving the performance of secure joint querying by two orders of magnitude compared to the implementation in previous work while also significantly extending its set of supported queries.

1 INTRODUCTION

Secure multiparty computation (SMPC) is an active research area that has recently gained popularity in securing the privacy of data. According to (Cramer et al., 2015), SMPC is a cryptographic technique that divides computing among several parties in order to ensure that no one party may see or infer the private information of other participants. This is different from traditional cryptography techniques in that it focuses on developing protocols for coordinating the processing of distributed data without joining the data, rather than data or databases. The main benefits of SMPC are that no third parties (no matter how trusted) see the data, the trade-off between data usability and data privacy is eliminated, and processing can be conducted with high accuracy. Today, SMPC is used for many real-life applications, such as detecting financial fraud, the aggregating model features across private datasets, and predicting heart disease (inpher.io,). Furthermore, it can help to solve such trust issues in contexts such as secure elections (Alwen et al., 2015), auctions (Aly and Van Vyve, 2016), and secret sharing (Evans et al., 2018). Thus far, in the context of databases, SMPC has mainly been used to secure relational databases such as Conclave (Volgushev et al., 2019). This raises the question of whether SMPC queries are restricted to relational databases or can transcend the database type. Against that background, in this paper, we explore the opportunity to apply SMPC to graph databases, a type of NoSQL database. Graph databases were created to address the limitations of relational databases (Salehnia, 2017), and they have found multiple applications in which the graph paradigm has been beneficial, such as on social media platforms (e.g. Instagram, Twitter, and Facebook) (Ciucanu and Lafourcade, 2020). In (Al-Juaid et al., 2022), we have proposed a design for secure multiparty graph databases. In this paper, we propose a fully automatic system to handle additional queries by using the Neo4j Fabric functionality extended with the Awesome Procedures on Cypher (APOC) library. Furthermore, we measure the execution times in experiments to validate our SMPQ system, and we review its overheads compared to Neo4j and Conclave.

The remainder of this paper is organised as follows: the following section 2 presents the background to this work, followed by the proposed approach in Section 3. Then, Section 4 is devoted to system im-

Al-Juaid, N., Lisitsa, A. and Schewe, S.
Secure Joint Querying Over Federated Graph Databases Utilising SMPC Protocols.
DOI: 10.5220/0011798900003405
In *Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP 2023)*, pages 210-217
ISBN: 978-989-758-624-8; ISSN: 2184-4356

plementation. Following this, Section 5 evaluates the performance of the system based on the results of our experiments. Next, the related literature is reviewed to understand how a query can be secured with different types of data models in section 6. Finally, the paper concludes in Section 7, and we offer directions for future work.

2 BACKGROUND

2.1 Secure Multi-Party Computation (SMPC)

SMPC allows a set of parties to jointly compute a mutually agreed function on their data while keeping their inputs private. Assume:

$$\mathsf{P} : \mathsf{P}_1, \mathsf{P}_2........., \mathsf{P}_n$$

$$\mathsf{X} : \mathsf{X}_1, \mathsf{X}_2........., \mathsf{X}_n$$

$$\mathsf{Y} = \mathsf{F}(\mathsf{X}_1, \mathsf{X}_2......., \mathsf{X}_n)$$

Where is P is a set of parties, each of them has a secret data X, and they want to compute some function F of their joint inputs. However, they do not trust each other and want to keep their input private. If the parties could agree on some trusted third party, Z, they would have to hand their data to Z, who would compute the function on their behalf and send them their prescribed part of the result Y. An SMPC protocol offers to compute F secretly with guarantees that the result will be the same as Z would get. Figure.1. shows an example of SMPC between 5 parties.

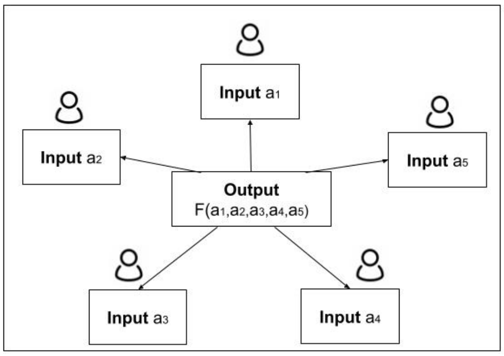

Figure 1: Multi-party computation between 5 parties.

Assuming that we have 5 parties, want to compute F using their private individual data without exposing it. SMPC helps to accomplish that with a guarantee of this.

2.2 Implementation of SMPC

SMPC was mainly the focus of theoretical study until recently, but there has been a significant effort lately to bring SMPC to applications in the real world (Evans et al., 2018). One of those implementations is JIFF(Albab et al., 2019).

JIFF is a JavaScript library that implements SMPC. It can carry out safe computing by utilising data that has been dispersed across several parts. JIFF uses a server to store and forward any encrypted messages sent between the various participants. It assumes the honest-but-curious security model and uses Shamir's secret sharing as the threshold (Shamir, 1979).

Each party runs as a client when JIFF is run as a server. SMPC operates by splitting each party's private information into smaller pieces—or shares— and then distributing those shares amongst several parties. Each share is useless alone, but when they are all combined, then the original secret is reconstructed. Once the computation finishes, the results are displayed for all parties without revealing their data used in the computation. The architecture of JIFF is illustrated in Figure.2.

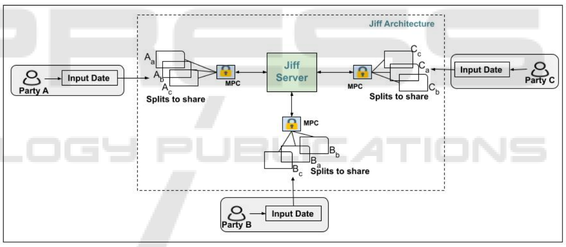

Figure 2: JIFF architecture and components.

As shown in Figure.2, when three parties, A, B, and C, want to compute something using their private data without exposing those to others after they connect to JIFF, the library will generate a JIFF client for each party to connect to the JIFF server. The input data from the parties are split into shares, distributed in encrypted form using the three parties' public keys (generated by the JIFF when all parties are connected) so that each party (client) holds one share for each of the other parties, as well as one of their own.

3 SYSTEM DESIGN

3.1 Overview

The architecture of our system is illustrated in Figure.3, in which multiple data owners, named

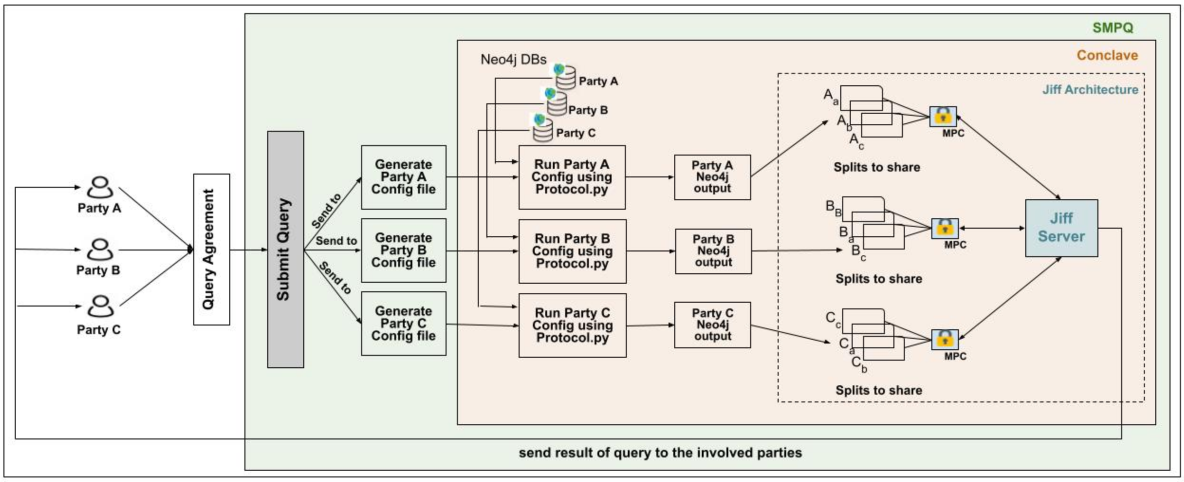

Figure 3: SMPQ workflow.

$p_1, p_2, \ldots, p_n$, own different graph databases but want to execute a single query jointly. To do so, firstly, all parties should agree on a joint query and then submit it to the system. After they submit the query, the system will automatically generate a configuration file for each party, containing the party's subquery and the information on their Neo4j database. Then, the Conclave system (Volgushev et al., 2019) runs these configuration files using the protocol, and by doing so, executes the query for each party. At this stage, the system passes the results of the query to the JIFF server to apply SMPC protocols. Within the JIFF server, SMPC splits each party's private information into smaller pieces—or shares—and then distributes those shares amongst several parties, as shown in Figure.2. In our current system, we use Conclave(Volgushev et al., 2019) for the backend, which in turn, uses JIFF (Albab et al., 2019) for SMPC queries. After the JIFF server finds the final result of the query, the system sends it out to the data owners who first initiated the joint query.

3.2 DB Query Language

Although any graph-based database can use our technique, we chose the Neo4j environment as our starting point. Neo4j is a popular graph database (López and De La Cruz, 2015), with a graph data model, which is presented as a collection of nodes representing data and arrows indicating the connections between them (Miller, 2013). Neo4j uses Cypher query language to deal with the data in such a graph(Francis et al., 2018). In our SMPQ system, the Cypher query language is then extended with the Neo4j Fabric functionality (Gu et al., 2022) and the APOC library (Needham and Hodler, 2019). The operational principle of Neo4j Fabric offers a way to issue Cypher queries that target more than one Neo4j graph database at once, which it implemented in federated databases. The APOC library, meanwhile, contains more than 450 procedures and functions to help with common tasks such as data integration, cleaning, and conversion, alongside general help functions. APOC is the standard library for Neo4j.

Our system's current functionality supports running multiple cypher queries, each in a separate database, extending with the Neo4j Fabric functionality, then applying an aggregation procedure from the APOC library. Furthermore, we tried slightly extending some of the APOC library's functionality. For example, the intersection procedure of APOC is supported to deal with two input sets, while in our system, we use the same syntax to express the intersection of three or more input sets.

4 SYSTEM IMPLEMENTATION

The SMPQ model was implemented using a Python API. It was built on top of the Conclave system (Volgushev et al., 2019); which uses SMPC with a relational database. Our model uses it as an interface between the data source and the implementation of SMPC protocols such as JIFF. This prototype used multiple graph databases from multiple data owners and applied one of the SMPC protocols (Evans et al., 2018) to provide a joint query. In practice, three data owners can perform a query jointly and can be adapted to be two or more data owners. Initially, three different Neo4j databases were built. In (Al-Juaid et al., 2022), we have proposed a system that was running manually from the start to generate the configuration files and pass them to Conclave (Volgushev et al., 2019) and run it. Currently, the functionality of our system is limited to the functionality that the Conclave system supports. To extend our work, we intend to increase the functionality to support all cypher queries that can be covered by the JIFF server extended with fabric architecture and apply the APOC procedures.

4.1 System Interface

The implementation of the proposed method interface can be observed in Figure. 4. As shown, after determining how many parties are involved in performing the query using SMPC protocol, they should agree on a computation ID which can be considered an agreement to apply the query using their database. When the *Query* button is pushed, behind the scene, the system will generate the configuration file for each party, pass it to the Conclave system to run using the JIFF server, and return the result as shown in Figure.4.

Figure 4: SMPQ system interface.

5 PERFORMANCE EVALUATION

5.1 Data Sets

To validate our proposed SMPQ system and investigate its efficiency, we ran several queries with three data sets, each from three parties that used different Neo4j databases. In the following, we describe each data set:

5.1.1 Professor and Students Data-Set

In this data-set, for all the databases, there were 58 nodes in total with 29 relationships between nodes. All three databases had the same nodes, *Professor* and *Student*, with the same two properties— *name* and *ID* —as well as an additional property for the Student—*score*.

5.1.2 Movie Data-Set

In the second data-set, we used the movie data-set after updating the nodes to add extra ones and remove others, to establish different sizes of parties. In total, there were 563 nodes, with 785 relationships between nodes. All three databases had the same nodes, *Actors* and *Movie*. The properties for node *Actors* were: name and born, while for node *Movie*, they were: tagline, title, and released.

Due to the Conclave system's limitations, which limit it to just numerical data, we employ a third data set that only contains numerical data to compare our system with the Conclave.

5.1.3 Car-Location Data-Set

In this data set, for all the databases, there were 100 nodes in total, with 63 relationships between nodes. All three databases had the same nodes *Car* and *Location*. The property for both node *Car* and *Location*: is the car_id, location_id, respectively. Table 1 shows further information about each data set, including the number of nodes and relationships for each database from different data owners.

Table 1: Information about all the three data-sets used to establish the experiments.

Data sets	Database	No of Nodes	No of Relationships
Profrssor-student Data-set	DB1	32	16
	DB2	10	5
	DB3	16	8
Movie Data-set	DB1	203	269
	DB2	172	254
	DB3	188	262
Car -location Data- set	DB1	44	31
	DB2	24	15
	DB3	32	17

5.2 Queries

We executed the following list of queries using the previous data sets.

- **Q1:** Count how many students there are in common between all DBs.
- **Q2:** Count how many students there are in common between all of the DBs that score 7.
- **Q3:** Find the names of the students there are in common between all the DBs with scores of 9 or above.
- **Q4:** Find the students' names in common between all the DBs that score 7.

- **Q5:** Count how many movies with the actor Tom Hanks there are in common between all DBs.
- **Q6:** Find the names of the movies in common with the actor Tom Hanks.
- **Q7:** Finds names of all actors born in 1974.
- **Q8:** Finds the sum of all nodes in the movie DB for all parties.
- **Q9:** Count how many cars are in each location.
- **Q10:** Count how many cars with ID = 1 in all the DBs.
- **Q11:** Union two databases using the node name of the Professors whose students have a grade $\geq$ 9.0 in either database.
- **Q12:** Project two databases using the scores for all students in both databases in the Math course.

5.3 Results

This subsection of the paper presents the results obtained when executing the above queries. We measured the execution times (i.e. the times taken for all parties to get the results of the query) using the *time* function supported by Python. Table 2 and Figure.5 highlight the execution times taken for all parties to get the results of Q1–Q8 using the previously mentioned datasets, with a comparison of the overheads of our system versus using Neo4j Fabric to run the same queries without SMPC protocols. As shown in the Table, the overheads are higher for our system; alternatively, using Neo4j Fabric to execute the query is clearly a better option, though our system offers greater security for the user since the data are encrypted using SMPC protocols. In the future, we intend to decrease the overheads of SMPQ by removing Conclave and instead directly connecting to the JIFF server to apply SMPC protocols.

Table 2: Execution times for Q1– Q8 when using SMPQ and Neo4j Fabric.

Query	SMPQ system party 1 (DB1)	party 2 (DB2)	party 3 (DB3)	Avg Time (s)	Neo4j Fabric Time (s)
Q1	1.52	1.49	1.49	1.5	0.136
Q2	1.95	1.93	1.94	1.94	0.147
Q3	2.02	2.02	2.01	2.01	0.161
Q4	1.84	1.85	1.85	1.84	0.132
Q5	1.78	1.79	1.74	1.77	0.182
Q6	3.05	2.96	3.08	3.03	0.012
Q7	2.74	2.76	2.77	2.75	0.154
Q8	0.87	0.86	0.83	0.85	0.278

When seeking to compare our system with the Conclave system, due to the limitation explained in (Al-Juaid et al., 2022) where Conclave supports only numerical data, we could not execute Q1–Q8. For

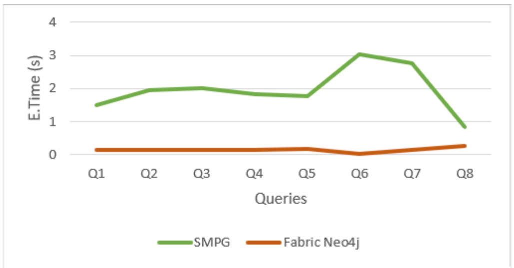

Figure 5: Execution times for Q1– Q8 when using SMPQ and Neo4j Fabric.

this reason, we used the third data set, which contained only numerical data, to run Q9–Q12. When running these queries using the Conclave system, the execution time took almost 10 minutes for a database with 100 nodes. We optimised the execution time for a query by removing the sorting function after finding the result for a query, which helped to reduce that time, bringing it down to 16 seconds. As further enhancements, we removed the waiting time until all parties were connected, and ran the queries for all parties simultaneously, which reduced the execution time to almost nearly 3 seconds. Table 3, and Table 4, and Figure.6 show the execution times for Q9–Q12 when using SMPQ and Conclave.

Table 3: Execution times for Q9– Q12 when using SMPQ.

Query	Execution time using SMPQ Party 1 (DB1)	Party2 (DB2)	Party3 (DB3)	Avg Time(s)
Q9	2.06	2.05	2.03	2.04
Q10	2.76	2.70	2.65	2.70
Q11	2.18	2.20	2.24	2.20
Q12	3.80	3.66	3.94	3.8

Table 4: Execution times for Q9– Q12 when using Conclave.

Query	Execution time using Conclave Party 1 (DB1)	Party2 (DB2)	Party3 (DB3)	Avg Time(s)
Q9	405.9	402.38	398.7	402.3
Q10	384.6	382.4	380.1	382.3
Q11	89.48	86.91	84.27	86.88
Q12	7.14	73.85	71.21	74.1

Table 5: 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	-	No	3	Relational DB	SQL	No	C
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
SMPQ	>= 2	Secret Sharing	JIFF	No	>= 2	GraphDB	Cypher	No	Python

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

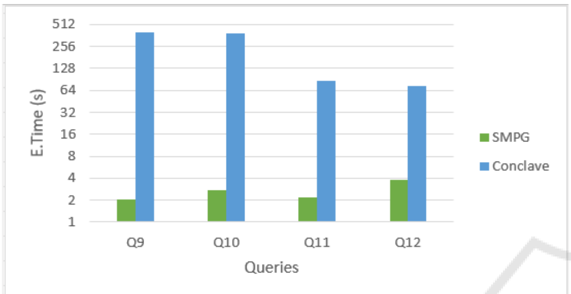

Figure 6: Execution times for Q9– Q12 when using SMPQ and Conclave.

6 RELATED WORK

6.1 SMPC for Data Processing

There are initiatives to utilize SMPC with databases to secure data. As an illustration, Conclave, a query compiler used with a relational database, is suggested by (Volgushev et al., 2019). It operates by converting the query into a series of short SMPC steps, local cleartext processing in data-parallel, and local processing. In this system, the queries are rewritten to reduce time-consuming SMPC processing and increase scalability. They suggested sending the revised query to JIFF, which serves as a backend SMPC system (Albab et al., 2019).

Additionally,(Poddar et al., 2020) presents the Senate system, which enables many participants to conduct joint analytical SQL queries without disclosing their personal information to one another. Their solution has the added benefit of offering protection against malevolent actors above earlier efforts, whereas those earlier systems employed a semi-honest architecture. Additionally, a relational SMPC system based on replicated secret sharing called Secrecy is presented by (Liagouris et al., 2021). The main idea behind this method is to divide the data into three parts, s1, s2, and s3, with each participant taking two of the shares and executing a portion of the query-running code.

SMCQL is a system suggested by the authors of (Bater et al., 2016) that converts SQL queries into secure multi-party computing. The user sends their query to a trustworthy broker who is thought to be honest. The query is translated to a secure cluster by an honest broker, who then returns the result to the user. A later study of the SMCQL system (Bater et al., 2020) adopts SMCQL and builds the SAQE system to secure the SQL query on top of it. In this system, the query is processed in two stages: the planning stage and the execution stage. The query plan and optimization are handled on the client side, and using SMPC on the server side, the query is executed among the data owners. They jointly run database queries and provide the client with the results. Similarly, Bater et al. construct on top of SMCQL system named Shrinkwarp (Bater et al., 2018). They conduct their studies using two data owners and perform two-party secure computations. SMCQL's performance has been increased, albeit at the cost of revealing some information in the process.

On the other hand, the authors in (He et al., 2015; Wong et al., 2014) and (Ciucanu and Lafourcade, 2020), suggested systems to illustrate how single-party querying can be implemented using SMPC. The SDB system in (He et al., 2015; Wong et al., 2014) is a cloud database system on relational tables. It has two parties: the server provider (SP) and the data owner (DO). Each item of sensitive data is divided into two shares: one held at the DO, known as the item key, and another at the SP, known as the ciphertext. In this system, the DO and SP share secrets using SMPC. When the SDB proxy in the DO part receives a SQL query from the user, it rewrites any queries with sensitive columns to their respective UDFs at SP. The rewritten queries are then sent to the SP, and the encrypted result is sent back to the SDB proxy for decryption before being sent to the user.

Likewise, the GOOSE framework in (Ciucanu and Lafourcade, 2020) is a solution that uses SMPC secret sharing to protect data outsourcing in the RDF graph database. Here, the data owner uploads the graph data to the cloud in a certain format: it is divided into three pieces and sent to separate locations in the cloud in encrypted form. All of these components are regarded as multi-party, and none of them can independently know the entire graph, a query, or its result. Additionally, the AES algorithm is used to encrypt every message sent between them.

Although SMPC has been used in relational databases and graph databases in the past, multi-party queries over graph databases are new. We, therefore, suggested a system called SMPQ. It helps to secure multi-party computation on graph databases. In order to conduct queries over graph databases, the SMPQ uses SMPC protocols. To show how well an SMPC query performed on a graph database, we implemented a prototype top on the Conclave system. Table 5 compares our proposed system, SMPQ, to all of the earlier systems.

7 CONCLUSION AND FUTURE WORK

In this paper, we have proposed a system for secure joint querying over federated graph databases based on secure multiparty computation (SMPC) protocols called SMPQ. We implemented our system using Conclave and enhanced a query's execution time until it was as close as possible to that of Neo4j Fabric. Furthermore, we expanded our system to be fully automatic and handle more queries than it did previously by using the Neo4j Fabric functionality extended with the APOC library. The current system remains to be tested on some Cypher query language, which was beyond the scope of this paper, such as a correlated query, and we have yet to add support for dealing with different databases separately. In future work, we will extend this system to handle traversal queries between all databases using SMPC protocols. Furthermore, we intend to reduce the overheads of SMPQ by removing Conclave and instead directly connecting to the JIFF server to apply 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.

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.

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.

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. (2022). A systematic overview of data federation systems.

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.

inpher.io. What is secret computing? https://inpher.io/technology/what-is-secure-multiparty-computation. Accessed: 2022-06-23.

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

López, F. M. S. and De La Cruz, E. G. S. (2015). Literature review about Neo4j graph database as a feasible alternative for replacing rdbms. *Industrial Data*, 18(2):135–139.

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.

Needham, M. and Hodler, A. E. (2019). *Graph algorithms: practical examples in Apache Spark and Neo4j*. Semantic Web.

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.

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

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

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.

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.