Performance trade-offs on a secure multi-party relational database

Rogério Pontes
HASLab, INESC TEC & University of Minho
rapontes@inesctec.pt

Mário Pinto
mariobpinto@gmail.com

Manuel Barbosa
HASLab INESC TEC & DCC FCUP
mbb@dcc.fc.up.pt

Ricardo Vilaça
HASLab, INESC TEC & University of Minho
rmvilaça@di.uminho.pt

Miguel Matos
HASLab, INESC TEC & University of Minho
miguelmatos@di.uminho.pt

Rui Oliveira
HASLab, INESC TEC & University of Minho
rco@di.uminho.pt

ABSTRACT

The privacy of information is an increasing concern of software applications users. This concern was caused by attacks to cloud services over the last few years, that have leaked confidential information such as passwords, emails and even private pictures. Once the information is leaked, the users and software applications are powerless to contain the spread of information and its misuse.

With databases as a central component of applications that store almost all of their data, they are one of the most common targets of attacks. However, typical deployments of databases do not leverage security mechanisms to stop attacks and do not apply cryptographic schemes to protect data. This issue has been tackled by multiple secure databases that provide trade-offs between security, query capabilities and performance. Despite providing stronger security guarantees, the proposed solutions still entrust their data to a single entity that can be corrupted or hacked. Secret sharing can solve this problem by dividing data in multiple secrets and storing each secret at a different location. The division is done in such a way that if one location is hacked, no information can be leaked. Depending on the protocols used to divide data, functions can be computed over this data through secure protocols that do not disclose information or actually know which values are being calculated.

We propose a SQL database prototype capable of offering a trade-off between security and query latency by using a different secure protocol. An evaluation of the protocols is also performed, showing that our most relaxed protocol has an improvement of 5% on the query latency time over the original protocol.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SAC 2017, April 03-07, 2017, Marrakech, Morocco

© 2017 ACM. ISBN 978-1-4503-4486-9/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3019612.3019659

CCS Concepts

•**Security and privacy** → **Data anonymization and sanitization; Management and querying of encrypted data;** •**Information systems** → *Query operators;*

Keywords

Secure databases, Derby, Multi-Party computation, Secret Sharing

1. INTRODUCTION

Cloud services are one of the best commodities of the modern age, with an increasing interest by not only the general population but also from private and governmental sectors. From the vast myriad of providers, an interesting set of opportunities are open to *outsource* storage and computation to services that have different degrees of scalability, availability, and reliability at an optimal benefit-cost ratio. On the act of *outsourcing* personal information to an external entity, one must be aware that this information most likely is not kept private and secure from prying eyes. In fact recent media scandals, regarding governmental surveillance programs, arose some trust issues regarding these cloud-based services.

This concern on cloud security is one of the main contributing factors that inhibits most enterprises from moving their databases to a cloud service despite the many benefits they would gain. In fact, bringing data privacy and secure query execution to databases is an open challenge proposed to the database community [1]. The challenge is being tackled by multiple solutions that apply different encryption techniques to ensure the security of the database. Considering only solutions based on software, the existing work can be divided into two main groups. The first group leverages multiple encryption techniques, where the security of these techniques is provided by computational hard problems. This same group, can for a subset of the SQL language provide secure queries. However to do so, all the data is stored on a single location and the encryption techniques used have different security guarantees [9, 10]. The other group of solutions is based on secret sharing schemes that provide data privacy by dividing data in multiple secrets stored at different locations and by using multi-party protocols to perform secure query computation [8, 11]. The main advantage of this group of techniques is the use of a

single encryption scheme that can be used to compute any function over the distributed data. Even when one site is attacked, the privacy of data is not broken. However, this security comes at a performance cost, with multiple messages being exchanged amongst the locations when computing a multi-party protocol [4].

We propose the first prototype of a Derby SQL database that leverages secret sharing and multi-party protocols to provide a more secure database capable of executing a subset of SQL queries. Furthermore, we offer a new trade-off between security and performance. Our protocol enables to fit the database performance to the users needs by slightly relaxing the security guarantees for this purpose. When relaxing the security guarantees of the protocols, security is only lowered when queries are executed, as the security of the stored data always remains the same.

1.1 Outline

Section 2 provides an overview on work developed not only on *Secure Multi-Party Computation* but also other approaches that intended to keep the data secure but still permit to carry out some sort of computation with it. In Section 3 the fundamental ideas of the *Secure Multi-Party Computation* and the *Sharemind* protocols are introduced. Section 4 presents our trade-off between security and performance 4. Section 5 presents the architecture and implementation details of a multi-party Derby database. Afterwards, in Section 6, there is an experimental setup, protocol evaluation and the discussion of the results. Section 7 concludes our work and provides prospects for future work.

2. RELATED WORK

CryptDB [9] is the first secure database solution that provides a comprehensive set of SQL queries that can be computed securely on an untrusted database. The main idea behind CryptDB is the application of different encryption schemes that provided trade-offs between security guarantees and the computation that can be made with the encrypted data. Prior solutions to CryptDB either provide a very small subset of SQL queries processed on the server side [12, 2] or a more complete set of SQL queries that require computation on the client side [5, 6].

Monomi [10] improves on CryptDB by actually doing some computation on the client side similar to previous solutions. However, the computation on the client is not strictly required and is only made when there is a significant performance improvement. The choice of the location for the query execution is made by a query *planner* that balances the execution of the query between the client and server. The improved performance of Monomi also comes from a *designer* component located at the client application that determines the optimal encryption scheme for the database tables based on samples of queries to be executed.

These solutions apply cryptographic schemes which security is based on computational hard problems and in some cases, they must relax the security guarantees to enable query capabilities on databases. Not only security guarantees have to be relaxed but data is often stored on a single untrusted domain. An alternative approach is given by solutions that employ secret sharing and secure multi-party protocols on databases. SDB [11] is such a solution, it uses two parties, where one party is the client and the other is a SQL database. Even though data is still stored on a single location, the database remains secure to an attacker that has access to the data stored on the database. This security guarantee comes from the use of secret sharing, where the secrets stored on the database are useless without the secrets kept by the client. SafeRegions [8] applies a different set of multi-party protocols to a NoSQL database. The protocols applied on SafeRegions are the same as the ones proposed by Sharemind [4]. Unlike SDB, the Sharemind protocols use three parties and do not require the client to keep any information or to participate on the query computation. Sharemind is also a data mining application that enables independent databases to perform analytical queries on data without having to disclose their private data [4]. Similiar to SafeRegions, our solution leverages the benefits of the Sharemind protocols and applies them to Derby, a SQL database.

3. SECURE MULTI-PARTY PROTOCOLS

Consider a model composed of a *dealer*, n independent *players* and an *attacker*. The dealer has sensitive information that must be stored safely by the players but no single player must know what information is being stored. The attacker's purpose is to steal sensitive information from the players. This model actually considers two types of attackers: an honest-but-curious attacker that can corrupt a single player, thus having access to every message exchanged but does not change any message; an active outsider attacker that listens and modifies the messages being exchanged. The system built from this model is secure against an honest-but-curious attacker by using *Secure Multi-Party Computation* and is secure against an active attacker by using secure communication between the entities.

For the dealer to be able to store its sensitive information securely, it can use secret sharing. Secret sharing is a cryptographic technique capable of dividing a value s in n secrets and assigns a secret to every player n. To restore the original value s from the secrets, a subset of the secrets must be sent back to the dealer. These schemes are secure as long as the attacker is only capable of accessing a predefined subset the secrets stored on the players.

While secret sharing can ensure the privacy of the stored data, it does not enable computation to be performed. The *Secure Multi-Party Computation* field of study aims to give a set of players a tool to securely compute a function without each player having to reveal information about their private knowledge. In a more formal light, suppose there is a game with n players, each one with its own private input x_n and the objective of the game is to compute a function f such that $f(x_1, \ldots, x_n) = (y_1, ..., y_n)$. In this game every player n can only know its own output y_n, none of them should be able to discover anything about the other players private data.

Our proposal builds a top of the protocols developed in Sharemind. As such the creation of secrets is done with additively secret-shared values on a cluster of three homogeneous machines. Any value $u \in \mathbb{Z}_{2^n}$ is divided in three secrets such that $u_1 + u_2 + u_3 \equiv u \mod 2^n$. A secret shared value is denoted by $[\![u]\!] = (u_1, u_2, u_3)$. Many protocols on the Sharemind framework use bitwise operations where a bit vector is represented by $\overline{u} \in (\mathbb{Z}_2)^n$.

The Sharemind computational model divides the interactive entities in three groups. One set of groups are the *Input parties* where each one generates the three shares for each of

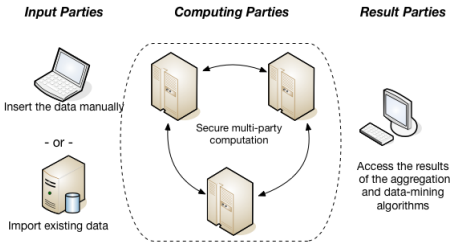

Figure 1: Sharemind architecture

its own input and sends them to the *Computing parties*. After the data has been stored in the *Computing parties* there is another group, the *Result parties* that require some computation to be performed and to be notified of the result.

From the many Sharemind protocols our work is focused on the **Equal(u,v)** and the **GreaterThan(u,v)** protocol, as they are fundamental operations in any Database engine. The protocols intricacies are not explained here in detail, instead we expose the main ideas and explain the security versus performance trade-off offered by our implementation. To give the reader some context on how the protocols work, we explain the addition protocol.

3.1 Addition Protocol

The addition protocol, adds two secret shared values without having to exchange any message between the parties. As presented in the Algorithm 1, each player holds two secrets, one secret from each value. The actual addition of the protocol is made by a local addition of each players' secrets. Afterwards, the protocol **Reshare** is used to ensure that the output shares are independent from the input shares. While the addition protocol does not require communication between the players, the **Reshare** protocol does.

Every Sharemind protocol follows this flow: the shares for the inputs are subject to local processing, possibly interleaved with secure communication with other parties. The final results of a computation are sent to the client by returning the individual shares via secure channels.

Algorithm 1 Protocol $[\![w]\!] \leftarrow$ Add($[\![u]\!]$,$[\![v]\!]$) for adding two secret shared values.

Require: Shared values $[\![u]\!]$ and $[\![v]\!]$.
Ensure: Shared value $[\![w']\!]$ such that $w' = u + v$
$\mathcal{P}_i$ computes $w_i = (u_i + v_i)(i = 2, 3)$
Return $[\![w']\!] \leftarrow Reshare([\![w]\!])$.

3.2 Original full security protocols

The two Sharemind protocols, **Equal(u,v)** described on Algorithm 2 and **Greater Than(u,v)** described on Algorithm 3 also follow a similar execution to the addition protocol. The main difference is each computing party's output bit. If the two satisfy the condition computed by the protocol, then the return value is one. Otherwise it is zero. Both protocols have local steps in the computing parties to calculate the difference between the values, but they also securely exchange messages between each other. The messages exchanged never reveal enough information for any computing party to independently figure out the inputs to the computation, nor its result. When applying these protocols to a database system they create a problem. If no computing party can figure out which values are indeed equal, or if u is greater than v, then the database is forced to securely compute and return the output bits for each row to the client. After the client receives every record of the database, it has to reconstruct the original values and discard the values that are not valid.

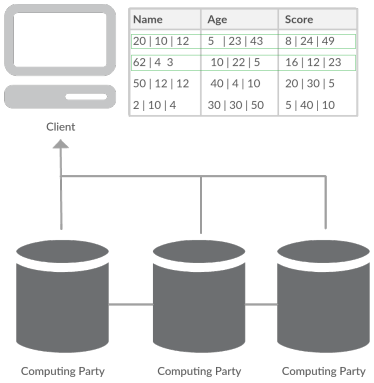

Figure 2: Sharemind protocols

Algorithm 2 Protocol $[\![w]\!] \leftarrow$ Equal($[\![u]\!]$,$[\![v]\!]$) for evaluating the equality predicate.

Require: Shared values $[\![u]\!]$ and $[\![v]\!]$.
Ensure: Shared value $[\![w]\!]$ such that w=1 if w=v, and 0 otherwise.
$\mathcal{P}_1$ generates random $r_2 \leftarrow \mathbb{Z}_{2^n}$.
$\mathcal{P}_1$ computes $r_3 \leftarrow (u_1 - v_1) - r_2$.
$\mathcal{P}_1$ sends r_i to $\mathcal{P}_i$ (i=2,3).
$\mathcal{P}_i$ computes $e_i = (u_i - v_i) + r_i (i = 2, 3)$.
$\mathcal{P}_1$ sets $\overline{\mathcal{P}_1} \leftarrow 2^n - 1 = 111 \ldots 1$.
$\mathcal{P}_1$ sets $\overline{\mathcal{P}_2} \leftarrow e_2$.
$\mathcal{P}_1$ sets $\overline{\mathcal{P}_3} \leftarrow (0 - e_2)$.
Return$[\![w]\!] \leftarrow BitConj(\overline{[\![p]\!]})$.

Algorithm 3 Protocol $[\![w]\!] \leftarrow$ GreaterThan($[\![u]\!]$,$[\![v]\!]$) for evaluating the inequality predicate.

Require: Shared values $[\![u]\!]$ and $[\![v]\!]$.
Ensure: Shared value $[\![w]\!]$ such that w=1 if $u > v$, and 0 otherwise.
$\mathcal{P}_i$ calculates $d_i \leftarrow u_i - v_i$
$[\![w']\!] \leftarrow ShiftRight([\![p]\!], 31)$
Return $[\![w]\!] \leftarrow Reshare([\![W]\!])$.

Figure 2 draws the scenario just described. The computing parties exchange messages and in the end they have to return every row and the output bits. In this figure the valid rows are represented by the green lines.

This strong security measure of not allowing the computing party to know which values are valid not only imposes an overhead on the messages sent to the client but also goes against the behavior expected from a database engine.

According to *D. Bogdanov* [3], the protocols, communication and round complexities are determined by the number of bits n of the data. The equal protocols requires up to $22n+6$ data bits and the GreaterThan uses $12(l+4) \times n+16$

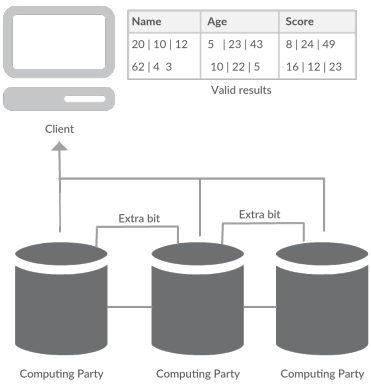

Figure 3: Sharemind protocols with relaxed security

data bits. Note that these values represent messages exchanged between the computing parties, and not the communication with the Client (this includes essentially one bit per row in the database to indicate the results).

The following section presents one possible trade-off that can be applied to both protocols, this trade-off results in a significant improvement in response time as shown by our evaluation. The performance improvement emerges just by sharing extra information between the parties. This extra information does not compromise in any way the original values.

4. PROTOCOL TRADE-OFF

The trade-off presented in this section works by revealing to the computing parties, through secure channels, just enough information to reduce the overall messages sent while not revealing the original values. The protocols exchange more messages among them but reduce the number of messages sent to the client. This leads to an improvement on the overall performance. At the same time, the information revealed does not allow any computing party to reconstruct the original values. However, an attacker can obtain additional information such as the users' access patterns.

4.1 Relaxed security

On each protocol, a compromise in security can be made to improve the protocols' performance. This is made by having the computing parties exchanging the resulting bits of a protocol. By gathering those bits, a computing party gains enough information to disclose the valid and invalid values of a query. Despite the increased number of exchanged messages between the parties, the messages sent to a client are reduced.

Figure 3 illustrates the trade-off of adding communication between the parties. However, the resulting table is only composed of valid rows. In fact, only six more messages are exchanged between the computing parties in each protocol. The number of messages sent to the client varies according to the stored values and the protocols, but, as showed by our experiments, this addition of messages ends up compensating with queries having lower latency. The new protocols are the same as the original until the return statement. Instead of returning the results of the protocols, additional computation is done as described in Algorithm 4. In the case of the Equal protocol, the new steps work on

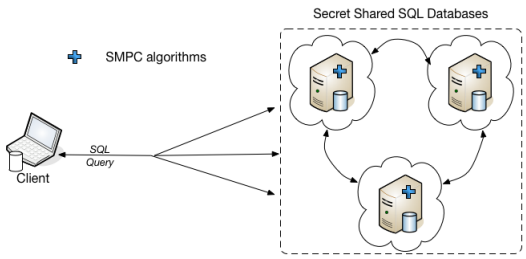

Figure 4: System architecture

the result of the BitConj protocol and on the case of the Greater than protocol the new steps work on the result of the Reshare protocol. These additional steps, share the resulting bits between the parties, thus enabling each computing party to know the output of the protocols and discard unwanted records. With the additional steps, the protocols either return 1 or 0 depending on the validity of the record.

Algorithm 4 Trade-off additional steps

$\mathcal{P}_1$ sends w_1 to $\mathcal{P}_i$ $(i = 2, 3)$
$\mathcal{P}_2$ sends w_2 to $\mathcal{P}_i$ $(i = 2, 3)$
$\mathcal{P}_3$ sends w_3 to $\mathcal{P}_i$ $(i = 2, 3)$
$w = (w_1 + w_2 + w_3) \mod 2$
Return w

5. ARCHITECTURE

The application of secret sharing and secure multi-party protocol to the Derby database requires a modification of its architecture and the addition of new components. As three parties are required to store the data and to perform multi-party protocols, our Derby database is in fact composed of three independent databases. Furthermore, each individual database contains a security library that handles the generation of secrets and computation of protocols. The last component added to the Derby database was a communication middleware that enables each party to exchange secrets and compute the protocols. A high-level conceptual view of the system architecture is depicted in Figure 4.

The database client also requires some modification to be able to not only communicate with the three database servers but also to generate secrets and decode the protocol results. In fact, this can be done with the same security library as the one used in the Derby databases.

Currently the only supported queries are SQL select statements with predicates that contain equal comparison or order comparison. Once a query with a equal or a order operator is issued, it is intercepted both at the client side and the server side. On the client side, the values being searched have to be encrypted with the secret sharing scheme. On the server's side, once the servers intercept the query a *Secure Multi-Party Computation* protocol is executed instead of executing the standard operation.

5.1 Implementation Details

The only queries being considered and used to evaluate the performance of the *Secure Multi-Party Computation* require a full scan of a database table. The Derby database has two main classes that are responsible to handle a full scan of tables. The first class is the *TableScanResultSet* that

given a database table and a Filter, it returns all the rows that satisfy the filter. This class however, only retrieves a single row from storage which has a negative impact on this system performance due to the overhead of reading a record from storage. The other class is the *BulkTableScanResultSet* which in fact is an extension of TableScanResultSet and reduces the overhead of reading records from storage by reading records in bulk.

The evaluation of the benchmark is focused on reading records in bulk and thus the class that is most used is the BulkTableScanResultSet. The adjustments to the Derby source code take action once this class does a fetch of the rows from the table. After the rows are loaded to memory, the filter to be applied to the database is intercepted. Once intercepted, the standard derby filter is replaced by a *Secure Multi-Party Computation* protocol takes place.

The *Secure Multi-Party Computation* protocols can also be applied to a single record or to a batch of records. To improve the performance of the systems, the security library that handles the protocols computations has in fact a slightly different implementation of the protocols presented as the implementation computes multiple secrets in batch and also sends this secrets to the communication middleware in batch. As the protocols require a high number of messages to be exchanged between the parties, it is advantageous to also send the messages between the parties in batch to lower the communication overhead.

The communication middleware handles all of the connections between the parties and abstracts the complexity of sending secrets from a single player to the others. A simple API is exposed by the middleware, that consists of two main primitives, *sendSecret(playerID, secret)* and *receiveSecret(playerID)*. Both of these primitives are also implemented to support the communication of batches of secrets.

The implementation of *Secure Multi-Party Computation* in a database imposes some limitations in its typical optimizations, making some of them impossible to use. Such an optimization is, for example, unique scans. With this optimization a selection query with a filter could be executed without the database having to scan the entire table. However, this capability is lost since shared secrets are not injective. Thus, one value can generate different values which makes it impossible to have a unique scan. This only penalizes even further the final performance of the database.

6. EXPERIMENTAL EVALUATION

The protocols' evaluation presented during this paper are implemented in a SQL database, so that we may not only assess their different performance trade-offs, but also have an idea of the cost of having a SQL database that executes its queries using *Secure Multi-Party Computation* protocols.

The experimental evaluation uses only a subset of SQL and a limited set of data types. The evaluated queries focus on Select queries with a Where clause that compares an equality or inequality of integers. The experiments were made on a single machine with an Intel Core i5 1,8GHz, 8GB 1600 MHz DDR3 of main memory and running the $OSX10.9.4(13E28)$. The database on which the *Secure Multi-Party Computation* protocols were developed was Apache Derby 10.10, with Java 1.8.0-20 using the SecureRandom Class with the interface SHA1PRNG from SUN to generate the random numbers. Every experiment used a batch

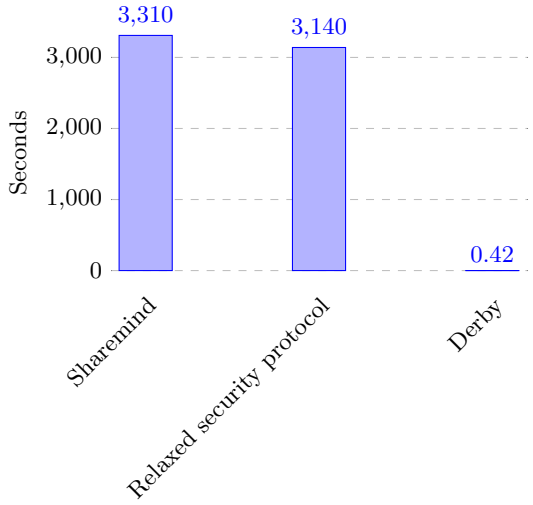

Figure 5: Equal protocol execution time

of 16 rows each time to minimize the number of rounds of communication.

The Equal protocol evaluation, used two SQL queries. These verify the equality or the difference of a value to every row in a similar manner to the queries in Listing 1.

Listing 1 Queries used to evaluate the equal protocol

```
SELECT * FROM table WHERE age = x
SELECT * FROM table WHERE age != x
```

The evaluation of the queries was carried out in a database with a table of 500,000 rows and three columns of integers. Each run measures the time between the client request and the answer arrival. Furthermore, it also contains the results' decryption latency.

Figure 5 contains the execution time for each version of the Equal protocol. The last vertical bar is the execution time of the standard Derby implementation while the first is the initial Sharemind protocol. As can be seen, there is a tremendous difference between the execution times of these versions. The original Sharemind protocol takes almost an hour to finish the computation and obtain the correct result while the default Derby implementation takes just 42 milliseconds to complete. The new protocols have an increase of performance of 5%.

The queries used to evaluate the greater than protocol can be found at the Listing 2. The conditions of the experiments are the same as in the equal protocol with a table of 500,000 rows and three columns of integers.

Listing 2 Queries used to evaluate the greater than protocol

```
SELECT * FROM table WHERE age > x
SELECT * FROM table WHERE age >= x
SELECT * FROM table WHERE age < x
SELECT * FROM table WHERE age <= x
```

The execution time of the most secure protocol is much higher than the standard Derby implementation and even than the equal protocol. The most secure protocol, the origi-

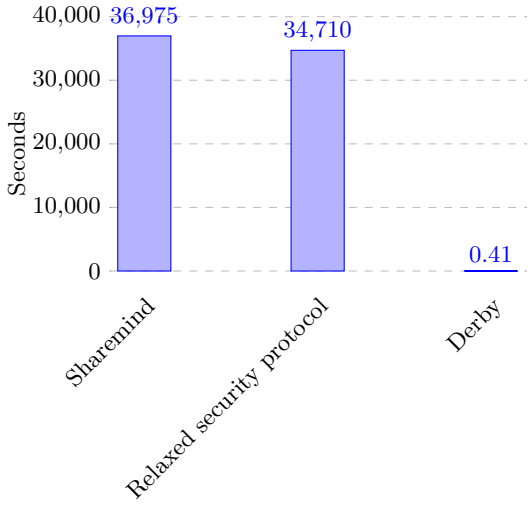

Figure 6: GreaterThan protocol execution time

nal Sharemind inequality protocol, takes more than 10 hours to successfully complete the computation. The relaxed security protocol provides an improvement of 6%, a difference of 2265 seconds. In fact, our protocol's performance is not constant but fluctuates depending on the dataset and the query. If a small number of values satisfy a query, then the query's latency also decreases as less values are sent to the client. However as the number of values that satisfy a query increases, so does the query latency. It is possible to reach a point where a query with our protocol can have a higher latency than the same query with the original protocol. This occurs when the overhead of messages in our protocol does not compensate the number of values sent to the client.

7. CONCLUSION AND FUTURE WORK

Sharemind protocols offer a set of algorithms capable of calculating equalities, inequalities, and arithmetic operations securely on values that are shared between a set of computing parties. However, when applied on relational databases, they impose a significant overhead in communication between the parties, leading to a real degrading in performance as our experiments demonstrated. However by diminishing the security of said protocols, it is possible to obtain efficiency increases of 5% without ever revealing the original values to any computing party. Nonetheless this improvement still take a significant time to compute simple queries. The query overheads might be acceptable only to certain applications, and certainly not to real time applications that requires almost immediate results.

Our implementation and evaluation consisted only on a subset of the data types and SQL operators that a relational database uses. This subset can be expanded to operations such as Join, Count, Max, Min and to support other data types such as floats as seen in recent research on these areas [7]. One other strand of research is creating new protocols with similar security to the ones presented here, or even with slightly lower security but that allow to have higher degrees of efficiency.

8. ACKNOWLEDGEMENTS

This work was partially funded by project "NORTE-01-0145-FEDER-000016", financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

9. REFERENCES

[1] Providing database as a service. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 29–, 2002.

[2] G. Amanatidis, A. Boldyreva, and A. O'Neill. *Provably-Secure Schemes for Basic Query Support in Outsourced Databases*. Springer Berlin Heidelberg, 2007.

[3] D. Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.

[4] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, pages 403–418, 2012.

[5] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. *Keep a Few: Outsourcing Data While Maintaining Confidentiality*. Springer Berlin Heidelberg, 2009.

[6] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 93–102.

[7] S. Laur, R. Talviste, and J. Willemson. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In *Applied Cryptography and Network Security*, volume 7954, pages 84–101. 2013.

[8] R. Pontes, F. Maia, J. Paulo, and R. Vilaça. SafeRegions: Performance evaluation of multi-party protocols on HBase. In *International Symposium on Reliable Distributed Systems*, 2016.

[9] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. pages 85–100, 2011.

[10] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *Proc. VLDB Endow.*, 6(5), Mar. 2013.

[11] W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, and S. M. Yiu. Secure query processing with data interoperability in a cloud database environment. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1395–1406, 2014.

[12] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving queries on encrypted data. In *Proceedings of the 11th European Conference on Research in Computer Security*, ESORICS'06, pages 479–495, Berlin, Heidelberg, 2006. Springer-Verlag.