FUDMA Journal of Sciences (FJS)
Maiden Edition
Vol. 1 No. 1, November, 2017, pp 49-57

SEQURE*SQL* – A FRAMEWORK FOR QUERY OPTIMIZATION AND PRIVACY ON OUTSOURCED DATA

[1*]Dima, R.M., [2]Obunadike, G.N.

Department of Mathematical Sciences and Information Technology,
Faculty of Science,
Federal University Dutsin-ma,
Dutsin-ma, Katsina State.
[1]*rdzer@fudutsinma.edu.ng*
[2]*gobunadike@fudutsinma.edu.ng*

Corresponding author:* **Dima, R.M. *rdzer@fudutsinma.edu.ng*

Abstract
Database outsourcing has become a trend in the information technology industry because it offers scalability to the enormous amount of digital content stored and generated on a daily basis by individuals and corporations. In large outsourced databases the efficiency of data retrieval, especially as it relates to privacy, remains an open challenge, because traditional query languages cannot work with encrypted data. While several architectures, techniques and tools have been proffered to ensure that privacy and performance are balanced and optimized, each of these approaches has its limitations. This research proposes a novel framework which focuses on optimizing server-side data retrieval and query efficiency through the use of hash map and AES 128-bit encryption algorithm. The design and implementation of secureSQL is built on the client-side without any alteration to the DBMS structure. SecureSQL model guarantees efficiency and is able to execute 20 out of the 22 Transaction Processing Performance Council (TPC-H) benchmark queries while ensuring privacy. This is proof that it is not restricted to simple query constructs but is able to handle even complex queries involving nested sub queries and joins. The execution time of queries between the client and database on the cloud is minimized, with a 0(1) time complexity as is evident in the comparative performance analysis between secureSQL and the traditional method. This is quantified using numerical results.

Keywords: *Cloud Database, Data Privacy, Hash Map, Query Processing, Encryption, DaaS, Relational Database.*

Introduction

Enterprises are becoming data-centric and increasingly producing huge amounts of data daily (Liu and Ting, 2010). Database outsourcing is a model that employs cloud computing technology to offer scalability, availability, broad network access and on-demand self-service among other obvious advantages to the digital content that is stored and transmitted to and from the storage servers. Cloud computing is an elevated form of grid computing, parallel or distributed computing which evolves to accelerate the idea of sharing resources expeditiously by offering access and the use of multiple server-based computational resources via digital networks where user may access the server resources using any kind of computing devices. A key problem in outsourcing the storage and processing of data is that part of the data may be sensitive, such as business secrets, credit card numbers, health records or other personal information (Kamara and Lauter, 2010).
Databases are traditionally protected by means of access control mechanism. This method guarantees security only when the data resides on a trusted server. Cloud database poses security risks to the sensitive contents due to possible malicious activities by internal and external parties. This challenge can be addressed by encrypting data before outsourcing in order to keep them hidden from the service provider (Davida *et. al*., 1981).

The storage and processing of encrypted data on storage servers is very desirable but implies a sacrifice of functionality for security (Song *et al*., 2000). Traditional encryption schemes prevent the execution of most SQL queries through the DBMS engine and conventional query languages cannot work with encrypted data.
In large outsourced databases, storing data is not problematic, retrieval is the challenge. Raybourn (2013) states that, the efficiency of data retrieval from untrusted servers especially as it relates to security, remains an open challenge. Before data encryption, one can send a precise query to the server and retrieve only the information needed. But to preserve the privacy of the data it is usually encrypted. Thus one cannot make selections on the server anymore because query languages cannot work with encrypted data thus limiting the computation. Therefore for each query there is need to download the database and do the decryption and querying on the computer before re-encrypting and uploading. Network connectivity and bandwidth problems further make the process tiring and cumbersome. The encryption key can be sent to the Cloud Service Provider (CSP) to do the decryption of the data, but this ends up in the same situation of vulnerability to snooping administrators, hackers, compromised servers.

Available query mechanisms reviewed, which allow server-side computations on encrypted data are limited as to nature

of data and queries they can execute. More so, security measures typically introduce significant computational overhead to the running time of general database operations due to cost of decryption which results in a potential disadvantage of performance degradation of queries. In essence, data retrieval is constrained. So the question arises: "What can be done to improve performance without trading off the security of a database system in the cloud?"

This research proposes a novel framework called secureSQL which aims at optimizing server-side data retrieval and query efficiency through the use of hash map and AES 128-bit encryption algorithm. SecureSQL model guarantees efficiency and is able to execute 20 out of the 22 Transaction Processing Performance Council (TPC-H) benchmark queries while ensuring privacy. The execution time of queries between the client and database on the cloud is minimized, with a 0(1) time complexity as is evident in the comparative performance analysis between secureSQL and the traditional method.

Review of Related Literature

The DaaS model was first introduced by Hacigumus *et al.,* (2002a). In his work, protecting the database records of a client from an untrusted database service provider was considered, by developing a model: NetDB2.

Querying data in encrypted form on an outsourced database focuses on devising techniques that allow computations on data that is encrypted using conventional encryption algorithms or the design and use of customized cryptographic schemes such as order-preserving or prefix-preserving encryption that allow computations to be performed on the data and they are applicable to numerical data or string data (Elmasri, 2008); but they do not provide a strong protection as they tend to leak a lot of information (Jacob, 2010). Generally, searchable encryption techniques such as the *probabilistic scheme* proposed by Song *et. al.* (2000) or the *fully homomorphic encryption* by Gentry (2009) tend to be computationally expensive and unable to scale as the size of the data increases.

Hacigumus *et al.* (2002b) explored techniques to execute SQL queries over encrypted data by processing as much of the query as possible at the server side without decryption and the remainder at the client side. This was done by using an algebraic framework (bucketization) for query rewriting to split (partition) the query for computation at both the server and client side thus minimizing computation at the client side by using metadata as index. Although privacy from the Service Provider was achieved with reasonable overhead, the technique employed could not handle aggregate functions like SUM, COUNT, AVG, MIN and MAX on encrypted numerical data. It was inapplicable for string data as well because the response of a query generated lots of additional data leading to high computational and communication overhead.

Popa *et al.* (2011), in their work: CryptDB implemented an approach that is built on top of the existing relational database management systems (RDBMS) and protects the privacy of database records in the cloud. It uses the concept of "onions". An onion consists of multiple layers of encryption schemes on a single data value before storing on the server. The removal of onion layer leads to performance problems. Any query which needs to remove a layer of onion runs slower when it is executed the first time because of the time taken to remove the onion layer.

Alhanjouri and Al Derawi (2012) proposed a mechanism to query encrypted data and make a tradeoff between the performance and the security by using the hash map data structure stored in the metadata component of the layer. The hash map stores the mapping between the plain text and the encrypted text as key:value pair. The work is implemented as a layer above the DBMS to manage the query process. The layer is placed on the same place with the database management system which is a limitation in the outsourced database scenario. Also, the encryption/decryption key is kept on the server side.

Sharma *et al*. (2013) did a research that aims to strike a balance between security and efficient query processing on encrypted databases by allowing users to query directly over the encrypted column without decrypting all the records thereby improving the performance of the system. They employ a technique which suggests two tables for a single main table. One table stores sensitive columns in encrypted format and the other table stores the encryption/decryption keys in encrypted format along with the content of the first table in plain text thereby hiding the relationship. Data retrieval is limited to less than 40% of the total. Also, the method is not suitable for the cloud environment.

In his dissertation, Kaul (2013) introduced a framework for solving the problem of executing queries efficiently at the cloud resident server while maintaining data security. PhantomDB maintains data security by using the onion method of Popa *et al.* (2011) which are carefully chosen to allow efficient query processing at the server. PhantomDB introduces the concept of Arithmetic Engine and Round Communication, which allow it to support all the standard SQL constructs with the exception of similarity operators. A hybrid storage model which takes unique features of PhantomDB into consideration while deciding the layout of relational data on disk is introduced to guarantee efficiency. A framework that is efficient without altering the structure of the DBMS will be preferable.

Bachhav *et al.,*(2017) gave a comprehensive survey on numerous models and approaches used for query optimization to minimize execution time and to improve resource utilization. They reviewed various research work done on query optimization for conventional SQL and MapReduce platforms.

It is pertinent to note that the problems considered in this research and the solutions proposed, differ from existing approaches. Since this research focuses on enhancing the performance and efficiency of queries on cloud databases with respect to time, the encryption scheme considered balances the tradeoff of security and performance through the use of selective attribute based encryption to secure sensitive data thus minimizing the amount of information represented in encrypted format. The preservation of data privacy is achieved through encryption. Query execution is achieved in almost constant time with the use of hash map and the application is designed for cloud databases. Also, the technique is integrated into the structure of the DBMS without any alterations to the storage model as in the case of Kaul (2013) as encryption is not expected to alter the database structure (Jacob, 2010).

Proposed Framework

The secure*SQL* system architecture is a breakdown of the major functionalities of the system. It employs the client-server architecture where the client is trusted and the server is not trusted. The trusted client layer is a customized application which is built using java applets to enable secured web browser access and is connected to the database which is outsourced to a public cloud server. The architecture is shown in figure 1.

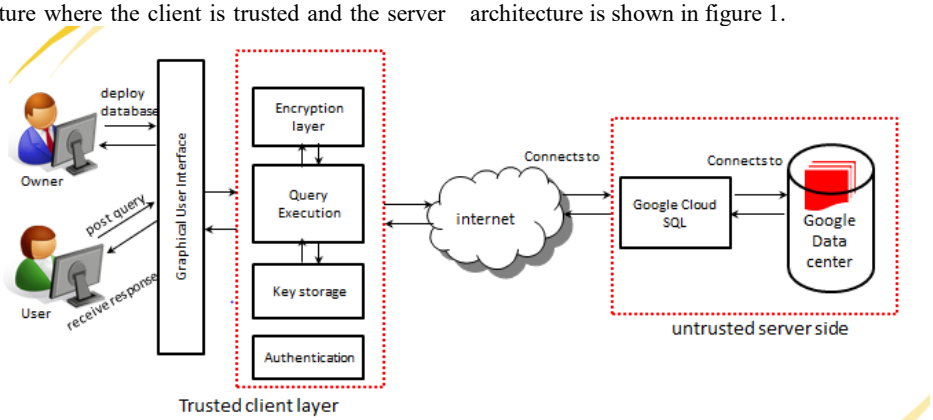

Figure 1: SecureSQL Framework

The encryption layer is expanded and shown in figure 2. The query execution process is also split into the various processing units and shown in figure 3. The encryption keys which are used for client side encryption and decryption of data are stored on the trusted client layer which is not accessible to the service provider. Authentication is used to identify the users of the system and ensure access control for the entities involved. The owner privileges and that of the user are clearly distinguished. A user should not encrypt or upload data to the database. The user can perform queries and receive results.

Encryption/Decryption Engine

The *batch encryption* captures the relation, attribute and primary key in other to encrypt data which is already stored in the resident database. *Single row encryption* is used to perform encryption update on records to be inserted into the database and this is illustrated in figure 2.

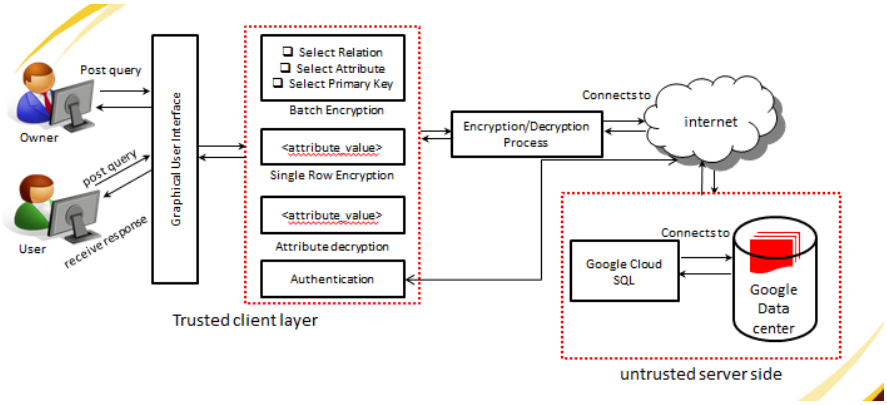

Figure 2: SecureSQL Encryption/Decryption Engine Architecture

The decryption process retrieves results which are still in encrypted form then decrypts on the client side and compiles the result (this is necessary because if the query involves both unencrypted attributes as well as encrypted attributes, the obtained result has to be compiled before it is displayed to the user).

In the decryption process, the angular braces "< >" act as a delimiter which is used by an inbuilt syntax analyzer. It indicates the beginning and end of an attribute that requires encryption to be applied to it. Hence, to query an encrypted attribute, the delimiter is used to group it so that the decryption function can be applied to it. Example: *SELECT <first_name>, last_name, <salary> FROM employees LIMIT 10;*

This statement indicates that the attributes first_name and salary are encrypted, thus the < > will decrypt the values and the result of first_name, last_name and salary records will be displayed in plaintext values.

Query Execution

A query analyzer is used to check for syntactic and semantic validity of the query. It checks that we have a valid sequence of tokens by breaking down a query into an array of tokens,

each containing a word, and then examining each token one character at a time using regular expressions.

The SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY tokens are analyzed one after the other.
Structures with recognizable patterns are then treated according to the required rule. The final string is a refined query ready for execution. Since the encryption is done according to selected fields, the syntax analyzer identifies which fields require decryption or further encryption by applying a set of rules to all columns enclosed in angular braces. This is done for each query posted. The query transformation relies on some metadata (the hash map) which stores a key *k* for secure hashing and indexing. Different columns use different keys and indexing expressions.
As shown in figure 3, users that need to access encrypted data submit their query which is then analyzed to know which attributes are to be accessed. Every attribute is associated with index information. Each plaintext query is mapped onto a corresponding query on the indexing content and executed in that form at the untrusted server. The hash map is initialized to fetch the key corresponding to the needed data then send the query to the cloud database for execution. Result of the query is returned to client for decryption and compilation and the final plaintext sent to the user. If the result is null, then a message is displayed to the user's interface as appropriate.

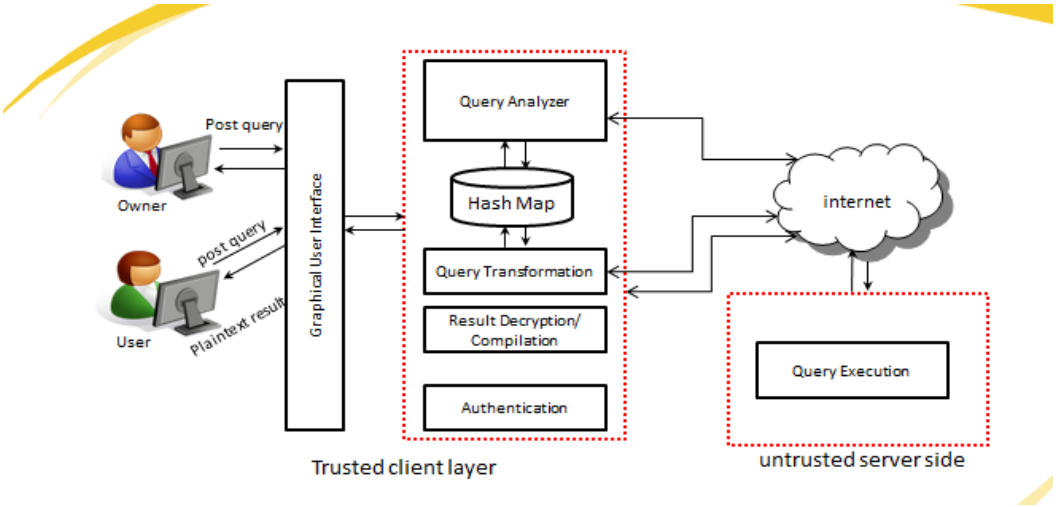

Figure 3: SecureSQL Query Execution Process

Users that only need to access the clear text content of the database submit the query directly to the server that stores the database and receive the clear text result with standard approaches.

S/No.	Algorithm
1	Start:
2	Define NewQuery as String
3	Define ValuesMap as Dictionary
4	LOOP (string Y in query.split(' '))
5	TEST: Y[0] == '<' and Y[Y.length-1] == '>' THEN
6	TEST: ValuesMap.HasKey(y) THEN
7	NewQuery += ValuesMap.GetValue(Y);
8	ELSE
9	NewQuery += AES_DECRYPT(Y);
10	ELSE
11	NewQuery += Y;
12	ENDLOOP
13	RETURN NewQuery;
14	End

Table 1: Query Execution Algorithm

Table 1 is an algorithm that explains the query execution process. ***Line 2*** declares a string variable to hold our query; ***line 3*** declares a dictionary to hold our key-value pairs in a hash map; ***line 4*** breaks query into bits using an empty space as delimiter; ***line 5*** determines if the column is to be encrypted or decrypted; ***line 6*** looks into the hash map first for the value; ***line 7*** if found, get value from the hash map instead of running a fresh decryption querying the database; ***line 9*** otherwise, run a new decryption for it; ***line 11*** if it is an ordinary column not requiring encryption or decryption; ***line 13*** new query returned for execution.

SecureSQL Design Specification

A provably secure encryption scheme – Advanced Encryption Standard which is implemented in a non-deterministic mode is used to ensure privacy of data. Performance with respect to queries is the key, even as the database is increased in scale, it is expected that the efficiency should not be compromised. Thus, hash map is used to achieve some constancy in time. The hashmap function $h(e)$ is used to store the encrypted values as key:

value pair by mapping the key e to a given range.

The implementation of hash map in this system ensures that the key:value pairs (plaintext:ciphertext) are stored in the metadata along with the schema of the database on the server. The hash map keys are values gotten from already existing data stored on the server, but when loaded on to the hash map at execution time, the keys become available in memory on the client side. Hash map is initialized in a millionth of a second which is quite negligible. It ensures that simple querying conditions do not require decrypting fields before being executed and this positively influences the time.

Example*: SELECT salary, last_name FROM employees WHERE id='12GT098';*
If *'id'* is the encrypted field, the query will not be executed. Thus, the hash map then serves as a lookup from which a replacement would be done at run-time by matching keys with corresponding values. So, as explained in the query execution process, the encrypted column is enclosed in angular braces < > which acts as a placeholder, and therefore treated as requiring decryption.

SELECT salary, last_name FROM employees WHERE id='<12GT098>';

Performance Evaluation

The experimental setup, consist of the google cloud SQL storage infrastructure with MySQL database. Data generated using TPC-H dbgen is used for the experiment. The primary keys and other schema dependencies are as defined by TPC-H. Application modules which incorporate the functionalities of the system architecture are implemented and they form the Client Side Application which is accessed via a Graphical User Interface.

Query Testing and Results

In order to test the performance, query response time was taken as a measurement across Data Manipulation Language (DML) statements with different conditions as expressed in the 22 TPC-H queries. Two main questions are of interest:

a. What is the scale up of the traditional method and the proposed method, that is, how does the query runtime vary with increasing dataset size?
b. How many of the TPC-H queries are actually able to execute on the proposed system?

Question 1 addresses scalability which is the trend of query execution time with increasing data size (varying workloads). Each statement was iterated at least 10 times and for every attempt query response time was noted and finally average was calculated for all iterations. The results for specific TPC-H benchmark queries (1, 2, 6 and 16) are presented. These specific queries contain all the characteristics exhibited by all the queries for string data (equality (=) and pattern matching (*LIKE*)) and numerical data (equality and range matching (<. >, =)). The number of records is plotted on the x axis and the execution time for

the different methods on the y axis.

TPC-H Query 1 captures a scan of the *lineitem* relation which contains 6 million records to retrieve data from nine columns involving computations of aggregate and range values with the *GROUP BY* and *ORDER BY* clause. Figure 4 is a line graph of mean execution time against varying number of records for query 1. Besides the wide variation in time between the two methods, the execution time for secureSQL is constant on the average despite the increasing number of records.

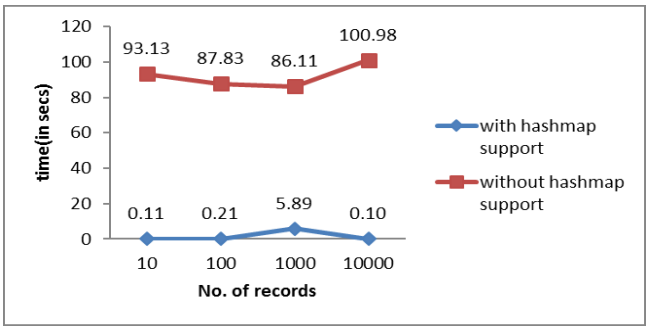

Figure 4: Graph of time vs no. of records for tpc-h query 1

TPC-H Query 2 entails a scan over five relations with nested sub-queries, equality tests and the *ORDER BY* clause to retrieve data from seven columns. Figure 5 is a line graph of mean execution time against varying number of records. It indicates that time increases linearly with the dataset - O(n), without the hashmap support and with the hash map support primarily because of the nested sub queries. SecureSQL performs 13.2% better than the traditional method.

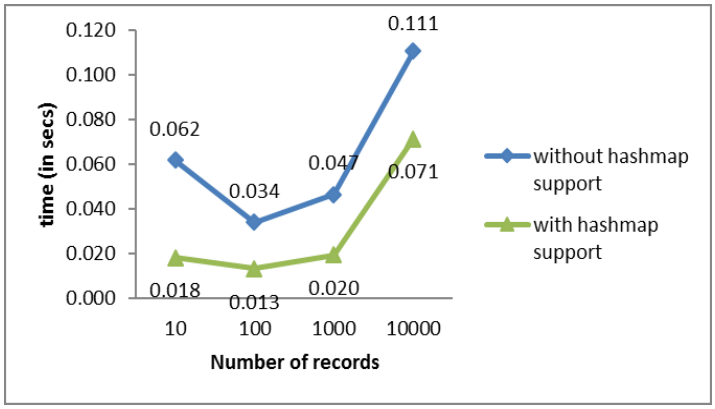

Figure 5: Graph of time vs no. of records for tpc-h query 2

Query 6 selects data from one relation involving one column and computes for aggregate values. Figure 6 is a line graph of mean execution time against varying number of records for query 6. SecureSQL query execution time is 0(1) despite increasing number of records.

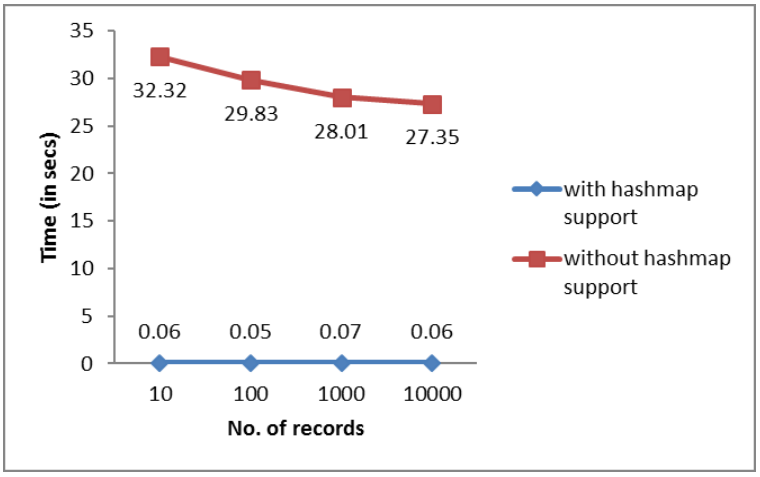

Figure 6: Graph of time vs no. of records for tpc-h query 6

Query 16 traverses two relations involving nested sub queries and computations of equality, pattern matching, *GROUP BY* and *ORDER BY* clause to retrieve data from four columns. Figure 7 is a line graph of mean execution time against varying number of records for query 16. SecureSQL query execution time is 0(1) despite increasing number of records.

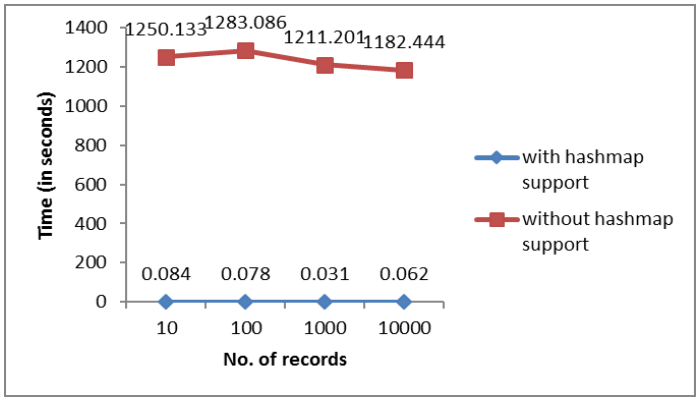

Figure 7: Graph of time vs no. of records for tpc-h query 16

Question 2 is focused on the query response time for different types of queries in the TPC-H benchmark and whether they can be successfully executed with the proposed method. The 22 queries were run and 20 executed successfully. Figure 8 is a linear graph plotted with query number on the x-axis and execution time on the y-axis using base10 log scale, for the raw execution time and the hash map supported method. . The wide margin between both methods necessitated the use of log scale on the time axis, in other to achieve a better visual rendition.

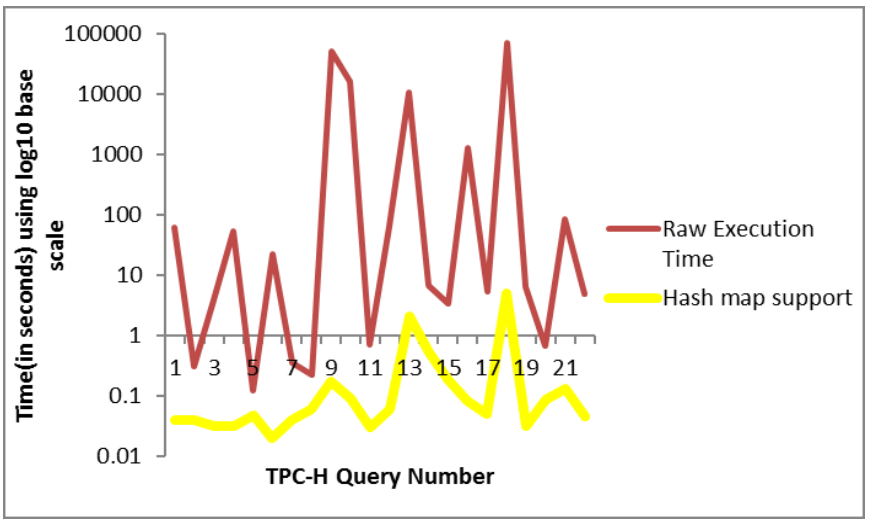

Figure 8: Graph of time vs queries using log scale

Observation and Discussion

On a general note, it can be observed that there is a significant variation between the execution time using the traditional method and the time using the proposed method. Also, with increasing number of records, the proposed method maintains a degree of constancy in time. Some exceptionally large figures were observed in the response time of few queries. This was usually the case when the query involved complex joins on multiple tables, *GROUP BY*, *HAVING* clause and nested sub queries. Resource utilization at the client side is another factor. Besides the exceptions, these modest overheads show that using the proposed method to query over encrypted data is practical in many cases with optimal performance.

Comparison of methods

Figure 9 is a chart which represents the number of TPC-H queries handled by other related methods compared. The proposed method handles 20 out of the 22 TPC-H queries as compared to Kaul (2013) – 16 TPC-H queries and Hacigumus et al., (2002b) – 2 TPC-H queries.

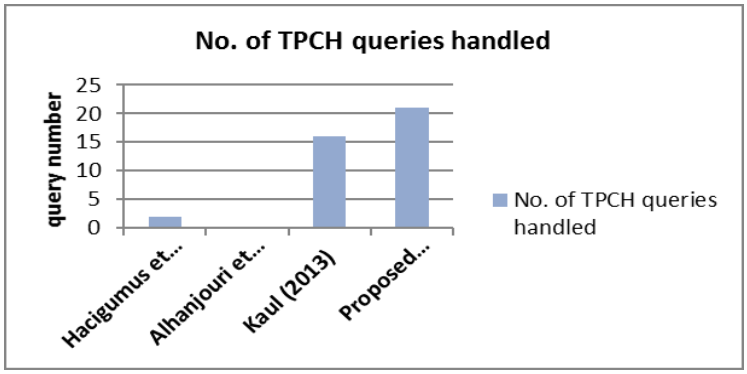

Figure 9: Comparison of Queries Handled

Summary and Conclusion

The efficiency of data retrieval in large outsourced databases, especially as it relates to the privacy of such data has been an open challenge primarily due to the fact that traditional query languages cannot work with encrypted data. Reviewed literature shows that most of the architectural models and encryption techniques proposed to ensure efficient performance of queries as well as privacy of outsourced data have limitations which range from high computational overhead to restricted query computation.
Based on these limitations, this research proposed an architectural framework which focused on optimizing server-side data retrieval and query efficiency through the use of hash map and AES 128-bit encryption algorithm. The implementation of this framework known as *secureSQL* is built on the client-side without any alteration to the DBMS structure. A query processing engine was also developed to process encrypted data. Selective attribute-based encryption which was incorporated into the encryption algorithm ensures that computational overhead is minimized.

Observation of the results obtained show that there is a significant variation between the execution time using the traditional method of decrypting entire records before performing queries and the time using the proposed method. Also, with increasing number of records, the proposed method maintains some degree of constancy in execution time. *SecureSQL* model guarantees efficiency and is able to execute 20 out of the 22 TPC-H benchmark queries while ensuring privacy.
In conclusion, the recent explosion of digital content ownership has both increased the popularity of data outsourcing and fueled concerns over data security. The need to facilitate storage and processing of large amounts and types of sensitive data is of particular importance in modern enterprise especially where the server is not trusted and client resources are limited. This research contributes in no small measure to eliminating the tradeoff between security and efficient query processing through its *secureSQL* design as is evidenced in the results.

References

Alhanjouri M. and Al Derawi A.(2012) "A New Method of Query over Encrypted Data in Database using Hash Map", *International Journal of Computer Applications (IJCA)*, **41**(4): 46-51, March 2012. Published by Foundation of Computer Science, NY, USA. Retrieved from: http://research.ijcaonline.org/volume41/number4/pxc3877580.pdf

Bachhav A., Kharat V. and Shelar M. (2017). Query Optimization for Databases in Cloud Environment: A Survey. In *International Journal of Database Theory and Application*. Vol.10, No.6 (2017), pp.1-12 http://dx.doi.org/10.14257/ijdta.2017.10.6.01

Davida G.I., Wells D.L. and Kam J.B. (1981). A Database Encryption System with Subkeys. *ACM Trans. Database Syst.,* **6**(2):312-328.

Elmasri R. (2008). *Fundamentals of database systems*. Pearson Education India.

Gentry C. (2009). *A fully homomorphic encryption scheme*. (Unpublished PhD thesis). Stanford University. Retrieved June 2, 2014 from: https://crypto.stanford.edu/craig/craig-thesis.pdf.

Gerardnico (2014, January 23). TPC - TPC-H Decision Support Benchmark Sample Schema [Web log post]. Accessed June 10, 2015 from: http://gerardnico.com/wiki/performance/tpc-h.

Hacıgümüş H., Iyer B. and Mehrotra S. (2004). Efficient execution of aggregation queries over encrypted relational databases. In *Proceedings of the 9th International Conference on Database Systems for Advanced Applications, Jeju Island, Korea.* Pages 633-650.

Hacıgümüş H., Iyer B., Li C. and Mehrotra S. (2002a). Providing Database as a Service. In *Proceedings of the 18th International Conference on Data Engineering* (*ICDE*'02). Sanjose CA. Pages 29-38. DOI: 10.1109/*ICDE*.2002.994695.

Hacıgümüş H., Iyer B., Li C. and Mehrotra S. (2002b). Executing SQL over encrypted data in the database service provider model. In *SIGMOD Conference*, pages 216–227.

Halitsch (2014). Implementation of TPC-H schema into MySQL DBMS [Web log post]. Accessed February 27, 2015 from: https://sites.google.com/site/halitsch88/Implementation-TPC-H-schema-into-MySQL-DBMS.

Jacob S. (2010). Cryptanalysis of a Fast Encryption Scheme for Databases and of its Variant. IACR Cryptology ePrint Archive 2010: 554. Retrieved June 15, 2015 from: https://eprint.iacr.org/2010/554.pdf

Kamara S. and Lauter K.(2010). Cryptographic cloud storage. In *RLCPS*, 2010

Kaul A. (2013). Query Processing in Encrypted Cloud Databases. (Unpublished Master's Thesis). Indian Institute of Science, Bangalore. Retrieved March 25, 2014 from: http://dsl.serc.iisc.ernet.in/publications/thesis/akshar.pdf.

Liu J. and Ting L.H. (2010). Dynamic Route Scheduling for optimization of Cloud Database. Presented at the *Intelligent Computing and Integrated Systems (ICISS)* pp. 680-682

Mykletun E. and Tsudik G. (2006). Aggregation Queries in the Database-as-a-Service Model. Retrieved August 20, 2014 from: http://www.ics.uci.edu/~gts/paps/mt06.pdf.

Popa R.A., Redfield C.M.S., Zeldovich N., and Balakrishnan H. (2011). CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP 2011)*, Cascais, Portugal, October 2011.

Raybourn T. (2013). Bucketization Techniques for encrypted databases: Quantifying the impact of Query Distributions.(Unpublished Master's Thesis). Bowling Greenstate University. Available December 2, 2014 from: https://etd.ohiolink.edu/ap/10?0::NO:10:P10_ACCESSION_NUM:bgsu1363638271.

Sharma M., Chaudhary A. and Kumar S. (2013). Query Processing and Performance and Searching over encrypted data by using an Efficient Algorithm. Retrieved on June 20, 2014 from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.303.6449.

Song D.X., Wagner D. and Perrig A. (2000). Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, *Berkeley, CA, USA*. pages 44-55.

Transaction Processing Performance Council (2014): Benchmarking TPC-H. Available from: http://www.tpc.org.