Balancing Query Performance and Security on Relational Cloud Database: An Architecture

A.A. Obiniyi
Department of Mathematics
Faculty of Science
Ahmadu Bello University, Zaria, Nigeria.

Rosemary M. Dzer
Department of Mathematics
Faculty of Science
Ahmadu Bello University, Zaria, Nigeria.

S.E. Abdullahi
Department of Mathematics
Faculty of Science
Ahmadu Bello University, Zaria, Nigeria.

ABSTRACT
Data 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. Of significance, is the security and privacy of the data outsourced to cloud servers when at rest and on transit. This challenge can be addressed by encrypting data before outsourcing in other to insure its privacy from internal and external intruders. Encryption implies a sacrifice of functionality for security. Querying encrypted data is performed by decrypting the entire data before retrieval. This leads to performance degradation. While several architectures, techniques and tools have been proffered to ensure that security and performance are balanced and optimized, each of these approaches has its limitations. We propose a novel framework based on the hash map principle, that enhances the security and performance of cloud database by reducing client SQL query response time on encrypted data without limiting the type of queries performed or compromising the security of data. Also, a performance analysis of the traditional method as compared with the proposed method is done using varying number of records.

General Terms
Security, Query Efficiency, Data Outsourcing

Keywords
Cloud Database, Security, Hash Map, Query Processing, Data Encryption, DaaS, Relational Database.

1. INTRODUCTION
A cloud database also referred to as Database-as-a-Service (DaaS) is a database that is accessible to clients from the cloud and delivered to users on demand via the internet from a cloud database provider's servers. It typically runs on a cloud computing platform such as Windows Azure, Amazon EC2, GoGrid, Google Cloud SQL [13]. The cloud platform can provide database as a specialized service or provide virtual machines to deploy any databases on.

Database services on the cloud are provided with automated features which enable optimized scaling, high availability, multi-tenancy and effective resource allocation. The services have the advantages of increased accessibility, automatic failover and fast automated recovery from failures, automated on-the-go scaling, load balancing, minimal investment and maintenance of in-house hardware, also better performance over the traditional database.

Databases are traditionally protected by means of access control mechanism. This method guarantees security only when the data resides on a trusted server. A critical question is: "to what extent is the confidentiality of sensitive data guaranteed once it is outsourced to a third party?" Cloud databases pose 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 [6].

The storage and processing of encrypted data on storage servers is very desirable but implies a sacrifice of functionality for security [3]. Traditional encryption schemes prevent the execution of most SQL queries through the DBMS engine.

1.1 Problem Definition
Some questions that arise in the area of research are:

a. what are the issues that affect performance of a cloud database with respect to client/database communication over the internet?

b. what is the performance in terms of a fully encrypted cloud database compared to a partially encrypted cloud database with the decryption keys stored on the client side?

c. what can be done to improve query performance without trading off the security of a database system in the cloud?

Analysis of cloud storage providers show that most but not all cloud storage providers offer built-in methods that encrypt the data to be stored in the cloud. However, the encryption schemes are mostly not sufficient, as some storage providers encrypt data by using an encryption key generated by and stored at the provider site. This implies that users cannot be sure of the confidentiality of their data [16].

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. To reduce this overhead, it should be possible to encrypt only sensitive data while keeping insensitive data unencrypted. Thus, only data of interest should be decrypted during query execution.

This research aims at minimizing the running time of client/cloud database queries without limiting the type of queries or compromising the security of the data.

1.2 Assumptions (Security Model)

a. Server is honest but curious (performs computations correctly but tries to glean additional information). This implies that the server may actively monitor all the data that it stores, as well as the queries it receives from the client, in the hope of breaching privacy; it does not, however, act maliciously by providing erroneous service to the client or by altering the stored data.

b. Client is fully trusted and won't be compromised hence stores the encryption/decryption keys. Preventing client-side breaches is a traditional security problem unrelated to privacy-preserving data storage, thus not covered within the scope of this work.

c. The communication channel between client and server is secure by using TLS (Transport Layer Security) and SSL(Secure Sockets Layer).

d. Server does not have any domain knowledge about the data being stored by client.

2. RELATED LITERATURE

This section contains an overview of current research trends related to the intended goal of this research.

Private database outsourcing model was first introduced by [9]. In his work, protecting the database records of a client from an untrusted database service provider was considered, by developing a model: NetDB2.

From a data access point-of-view, dealing with encrypted information represents a burden since encryption makes it not always possible to efficiently execute queries and evaluate conditions over the data. Straightforward approach is to encrypt all the data as in the data outsourcing scenario [4][9].The assumption for such approach is that all the data are equally sensitive; therefore, encryption is the price to protect them. This is overkill because the sensitivity lies in the data association with other data.

Following his pioneer work, [8] 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 for query rewriting to split 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 cannot handle aggregate functions like SUM, COUNT, AVG, MIN and MAX on encrypted numerical data. Also, applying it to string data that has a wide range is inapplicable because a lot of additional data is sent to the response of a query leading to high computational and communication overhead.

To overcome the limitation of the previous works, [7] used privacy homomorphism to support aggregation queries on outsourced data.

A new mechanism was proposed by [14] to query encrypted data and make a tradeoff between the performance and the security. The work was implemented as a layer above the DBMS to manage the query process. The method used was based on replacing the select conditions on the encrypted data with indexing information about the data which should be related with the data well enough to provide an effective and faster query execution mechanism. By using a one-to-one function that generates a new value for the original data attackers don't guess the original input value from the output.

A shortfall in this work is that the layer is placed on the same place with the database management system. This cannot work in an untrusted environment. Also, the encryption/decryption key is kept on the server side.

In continuation of the previous research, [15] did some modifications to the layer technique by using a data structure called hash map 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. A well-dimensioned hash map ensures that the average cost for each lookup is independent of the number of elements stored in the table.

Aiming at striking a balance between security and efficient query processing on encrypted databases, [12] implemented a technique that allows users to query directly over the encrypted column without decrypting all the records thereby improving the performance of the system. They employed 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. The result of this technique is satisfactory only when data retrieval is less than 40% of total. In a transaction processing environment this is not satisfactory.

A framework for solving the problem of executing queries efficiently at the cloud resident server while maintaining data security was introduced by [1]. His work, PhantomDB maintains data security by encrypting the data under multiple encryption schemes before storing it on the server. The Encryption schemes used are carefully chosen to allow efficient query processing at the server. He introduces the concept of Arithmetic Engine and Round Communication, which allows it to support all the standard SQL constructs with the exception of similarity operators. A hybrid storage model which takes unique features of the system into consideration while deciding the layout of relational data on disk is introduced. The framework performs efficiently only with the custom storage model. A framework that is efficient without altering the structure of the DBMS will be preferable.

3. SYSTEM MODEL

As depicted in Fig. 1, the system model is made up of two major components. The client layer which is trusted, and the server side which is not trusted. The client layer consists of the application which is accessed through a web browser. It connects through the internet to the cloud service provider (CSP) and then to the data center, both of which make up the server side.

The main functions performed are:

a. Deploying a database

b. Posting and processing queries

The main entities in the system model are:

a. Data owner: A person or organization who produces and outsources encrypted data to a server and also provides some additional information called metadata to client side for the purpose of query transformation.

b. Client: A trusted front-end entity that translates the queries posed into equivalent one for processing on encrypted data at server side.

c. The user: An entity that requires to access data (encrypted or unencrypted) stored at server by querying through a client front-end entity.

d. The cloud service provider (CSP) or Server: An external entity that returns the response of query to the client who in turn decrypts and returns the result to the user.

3.1 Dataflow in the Model

The data owner/client deploys his relational database with some attributes encrypted and some attributes in plain text using the client side application to the cloud.

The user posts a plaintext query which is executed on the server based on whether it is accessing encrypted attributes or not. If the access is on encrypted attributes, the result is returned encrypted from the server and decrypted on the client side. The plaintext result is then displayed to the user. If the result is null, then a message is displayed to the user's interface as appropriate.

3.2 Background of Model

3.2.1 Hash Map

HashMap is derived from hashtable. It is formed by transforming a range of key values to a range of array index values by using a hash function. A hashmap could be implemented using Closed hashing (open addressing) or Open hashing (closed addressing). The latter will be used in the implementation of this work. Open hashing is a large array of O(n) elements, called buckets. Each bucket is a set containing O(1) elements and the elements of the set as a whole are partitioned among all the buckets. This ensures optimal performance especially when the hash function and resizing threshold are tuned to determine the load factor $\propto$.

To determine the bucket that stores the element, use a hash function h(*e*) that given a set element *e* returns the index of a bucket that should contain that element. Therefore, h(*n*)=*n* mod *m* for *n* number of elements (keys) and *m* number of buckets.

In a well dimensioned hashmap, the map size is proportional to the number of entries O(*n*) and the average cost (number of instructions) for each lookup is independent of the number of elements stored in the table. The time complexity denoted by big O for average case is O(1) for the search, insert and delete operations.

If we have *n* elements in the set and the bucket array is length *m*, then we expect the load factor, $\propto = {}^{n}/_{m}$ element per bucket. We ensure that the load factor $\propto$ never exceeds $\propto_{max}$ where max = 0.75 according to java's hashmap class recommendation. So, all operations are O(1) on average and collisions are avoided.

The hashmap function h(*x*) is used to store the encrypted values as key:value pair by mapping the key *x* to a given range. This is kept in the metadata along with the schema of the database. Hashing randomizes the data order and provides efficient access to information based on the key.

3.3 Architecture

The system architecture is a breakdown of the major system components into their various functionalities as shown in Fig. 2. The trusted client layer which is built using java applets to enable secured web browser access, contains the encryption/decryption engine, the metadata and schema of the database as well as the query analyzer. It also stores the decryption keys and performs authentication where necessary.

The encryption/decryption engine is responsible for encrypting the selected attributes using the AES 128-bit algorithm in CBC mode before deploying to the server. The hash map function which stores the encrypted values as key:value pair is contained in the metadata along with the schema of the database.

3.3.1 Encryption/Decryption Engine Architecture

In Fig 3, queries posted for computation/processing are analyzed by breaking down into query components/blocks as follows:

a. Select
b. From
c. Where
d. Group By
e. Having
f. Order By

These blocks are checked whether they require access to one or more encrypted columns. If so, the user is authenticated before being granted or denied access.

All the keys are stored in the encryption engine on the client side as well such that, they are used for decryption of cipher text on the server without granting access to the service provider.

Google infrastructure will be used to implement and test the system because of its obvious advantages. The google engine will be used for storing the application and the google cloud SQL for storing the relational (SQL) database.

3.3.2 Query Execution Process

In Fig. 4, when the owner is uploading or updating data, the application interface allows him to select the entity or relation he wants to encrypt. The attributes contained in the relation are displayed and he selects which attributes to encrypt and the encryption function to use. The data is then deployed to the google SQL database via the google app engine.

The system randomizes the attribute names so that there is no correlation between the cipher text and plain text. Retrieved results which are still in encrypted form are decrypted and the result compiled (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).

The query execution is also done by the encryption/decryption engine. A query posted by a client is checked to know if it involves one or more encrypted attributes. If so, query translation takes place whereby some metadata is obtained about the attribute(s) to enable query computation at the server. This metadata contains the key:value pair hash map, the information about the randomized values and any other information needed by the client.

4. IMPLEMENTATION

The system will be implemented using AES 128-bit in CBC mode in java to encrypt pre-selected columns. The CBC mode is a non-deterministic encryption scheme. This means that encryption of same value multiple times will give different cipher texts with a very high probability, without the need for a slower re-keying process. It is achieved by using

randomization during encryption of data. The decryption key is created and kept on the client side. An index is built over encrypted column to make the searching faster giving a constant lookup. By finding the needed encrypted value, we find the needed plain text. This is done by using the same algorithm with the same symmetric key.

4.1 Performance Analysis

To show the validity and efficiency of our proposed approach, simulation will be done with the TPC-H benchmark database on a PC with windows 8 platform, 64-bit OS with intel(R) core i5 CPU at 2.50GHz, 8GB RAM. Experiment will be performed by measuring the response time of queries over fully encrypted data and partially encrypted data with varying number of records.

5. SUMMARY AND CONCLUSION

The proposed architecture should work with all data types, that is, character, numeric and others. It should also allow all SQL query types to be executed. It is also expected that the speed of query execution should remain minimal and constant despite the increasing number of records. In future, the architecture needs to be transformed to a practical and workable design in other to verify the validity of the expected results. Also, introducing client side caching to speed up cloud/client queries is another interesting research direction.

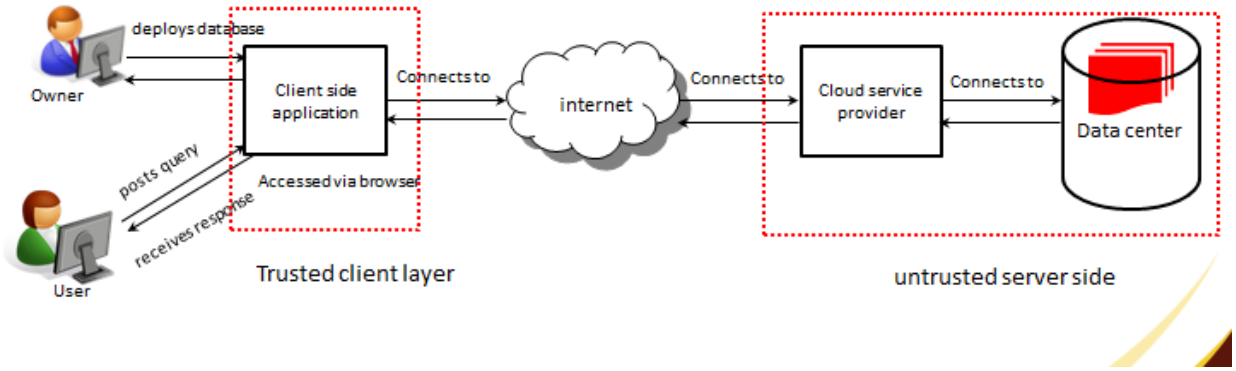

Fig 1: Overview of the system model

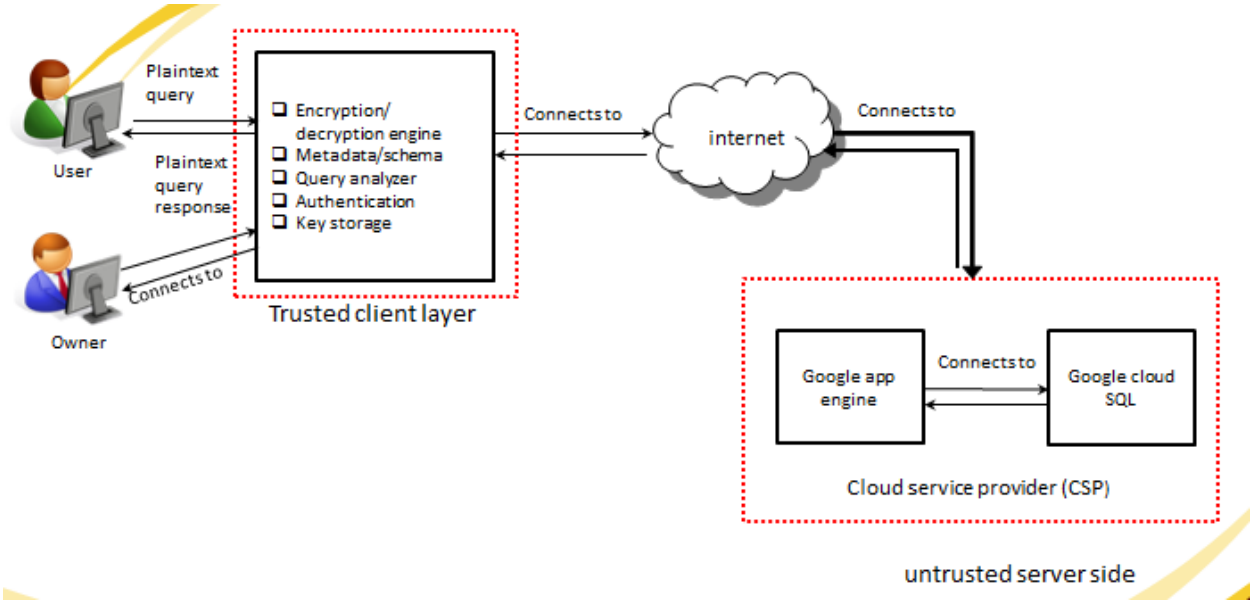

Fig 2: System Architecture

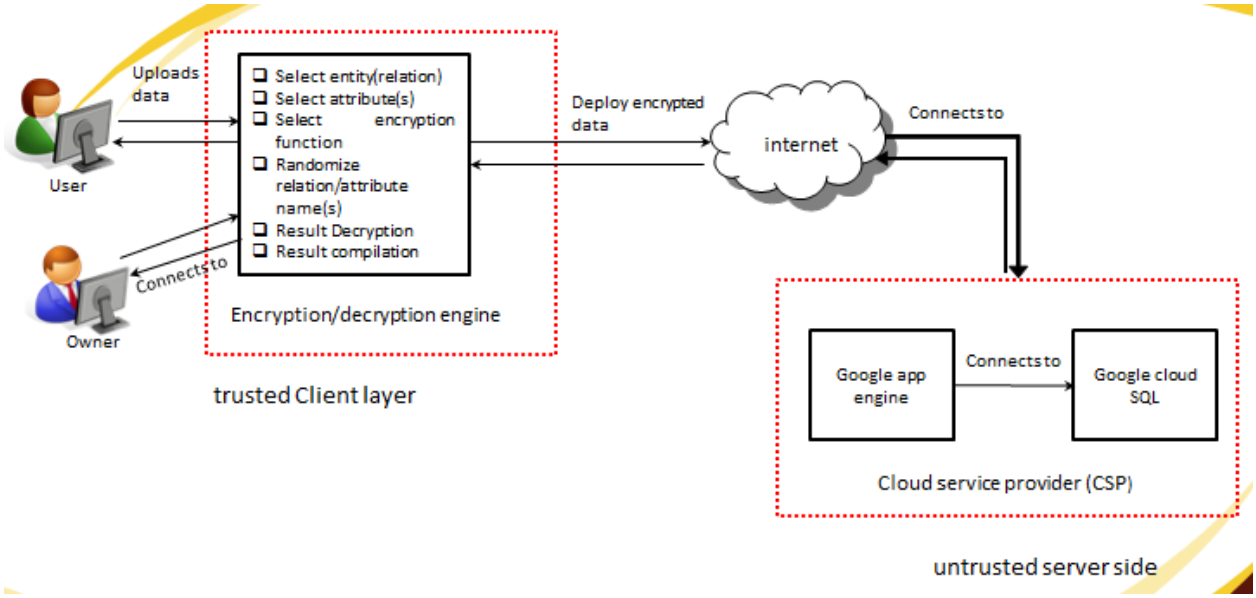

Fig 3: Encryption/Decryption Engine Architecture

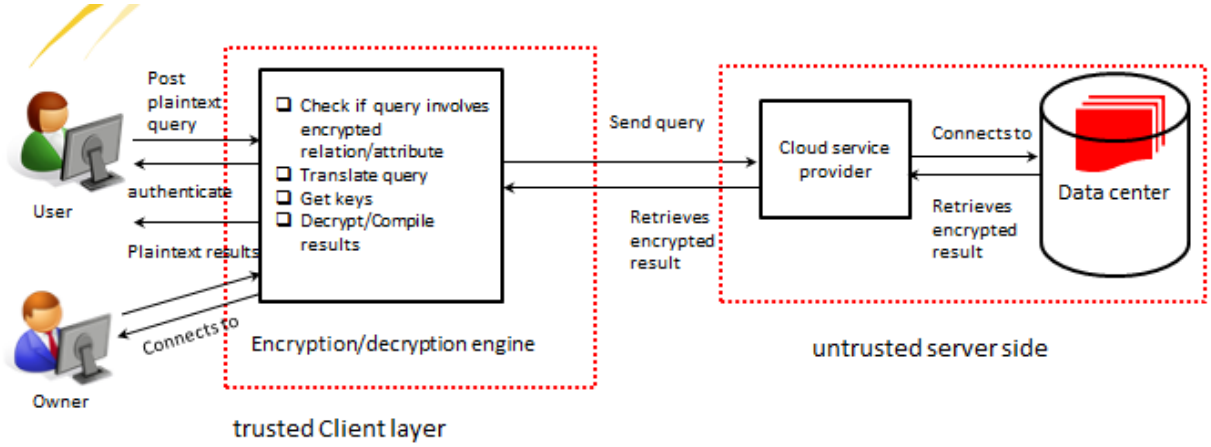

Fig 4: Query Execution Process

6. ACKNOWLEDGMENTS

Our thanks to the Almighty God, who made this work possible.

7. REFERENCES

[1] Akshar Kaul 2013. Query Processing in Encrypted Cloud Databases. A Master's Thesis. July 2013.

[2] Brinkman R. 2007. Ph.D Thesis. Searching in Encrypted Data.

[3] Dawn Xiaodong Song, David Wagner, Adrian Perrig (2000). Practical techniques for searches on encrypted data. In IEEE Symposium on Security and Privacy, pages 44-55.

[4] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati 2003. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS), pp. 93-102, 2003*.

[5] Frakes B. William and Baeza-Yates Ricardo 1992. Information Retrieval Data Structures and Algorithms. A Book

[6] George I. Davida, David L. Wells, John B. Kam 1981. A database encryption system with subkeys. *ACM Trans. Database Syst.*, 6(2):312-328.

[7] Hacıgümüş, H., Iyer, B., and Mehrotra, S. 2004. Efficient execution of aggregation queries over encrypted relational databases 633-650. In Proc. of the 9th International Conference on Database Systems for Advanced Applications, Jeju Island, Korea, March 2004.

[8] Hacıgümüş, H., Iyer, B., Li Chen and Mehrotra, S. 2002b. Executing SQL over encrypted data in the database service provider model. In SIGMOD Conference, 2002.

[9] Hacıgümüş, H., Iyer, B., Li Chen and Mehrotra, S., 2002a. Providing Database as a Service. In Proceedings of ICDE, 2002.

[10] What is Google Cloud SQL? https://developers.google.com/cloud-sql/docs/. Accessed on: January 21, 2015.

[11] Lafore Robert 1999. Teach yourself data structures and algorithms in 24 hours. SAMS publishing. A Book

[12] Manish Sharma, Atul Chaudhary and Santosh Kumar 2013. Query Processing and Performance and Searching over encrypted data by using an Efficient Algorithm. A Paper.

[13] Mateljan Vladimir, Cisic D., Ogrizovic D. 2010. Cloud Database-as-a-Service(DaaS). ROI. In MIPRO, 2010 proceedings of the 33rd International convention, pages 1185-1188. IEEE

[14] Mohammed Alhanjouri and Ayman M. Al Derawi 2011. New Method of Query over Encrypted Data in Database. A Paper.

[15] Mohammed Alhanjouri and Ayman M. Al Derawi 2012. A New Method of Query over Encrypted Data in Database using Hash Map. A Paper.

[16] Montz Borgmann, Tobias Hahn, Michael Herfert, Thomas Kunz, Marcel Richter, Ursula Viebeg, Sven Vowe 2012. Sit Technical Reports: On the Security of Cloud Storage Services. Fraunhofer Institute for Secure Information Technology SIT, Germany. March 2012.