Journal of Computer Science

Original Research Paper

An Efficient Approach for Query Processing Over Encrypted Database

Jaafer Al-Saraireh

Department of Computer Science, Princess Sumaya University for Technology, Amman, Jordan

Article history
Received: 16-04-2017
Revised: 2-10-2017
Accepted: 13-10-2017

Email: j.saraireh@psut.edu.jo

Abstract: A novel approach proposes in this research to improve the performance of the query over encrypted database. This approach based on using hash map function to generate a unique hash value for each and every sensitive data. In this a proposed approach, there are no relationship between the hashed value and encrypted value. This method can reduce the cost of the encryption and decryption operations and improve the query performance. Results of the set of experiments show that the query performance over encrypted data reduces the response time to 25s, 51s, 34s, 61s and 31s for queries 1, 2, 3,4 and 5; respectively. The comparison with other methods are carried out when database size is ranging from 50,000 to 100,000 rows.

Keywords: SQL, Database Security, Encryption, Hash Map

Introduction

Database is a collection of related data, data is a fact can be recorded and have implicit meaning. Data can be classified as sensitive data and insensitive data. The sensitive data is protected in database by using encryption mechanisms. The encryption mechanisms can be classified as symmetric and asymmetric encryption. In symmetric encryption technique the single or shared key is used to provide the confidentiality service in database; while in asymmetric encryption technique two keys, public and private key are used to provide the security.

The tradition security polices such as access control, physical security and network security don't sufficiently provide a secure support for storing and processing sensitive data in a secure way. The encryption technique provides an efficient method to store sensitive data in secure way (Rathod and Dhote, 2014; Nassar *et al*., 2017; Ali and Afzal, 2017).

Cryptographic has been widely used to support database security. However, as in any case where information security was addressed, performance directly affected. The cost of encryption and decryption data being inserted or retrieved from database adds to the regular cost of storing and retrieving data from an unencrypted database.

The traditional method to retrieve an encrypted data is to decrypt all the sensitive data to plain text then find the target records. This method obviously cost time and has a bad performance especially with a large number of records.

This work addresses the problem of securing data in database while preserving the performance of the database system without major degradation. We explore the possibility of improving on existing work by proposing an approach that combines security with performance and can be used for different types of databases.

Therefore; a new proposing method to query encrypted sensitive data. The propose method has a good comparable response time with the traditional method.

Literature Review

The factors related to encrypting databases to protect them against attacks was presented by (Bouganim and Guo, 2011). The first factor was related to the levels of encryption was performed. The researchers identified three levels of encryption were performed on storage-level, database-level and application level. The next factor was related to the encryption algorithm used, e.g., DES and AES and the key management methods involved in the respective algorithm.

Salama *et al*. (2010) evaluated set of encryption algorithms by studying their effects on the performance of the involved systems from the viewpoint of different data block size, different data types, battery power consumption, different key size and encryption/decryption speed.

The evaluated algorithms are AES (Rijndael), DES, 3DES, RC2, Blowfish and RC6. The experiments conducted lead to conclude that AES leads in terms of

speed while Blowfish is performing better when packet size is varied. Meanwhile, the type of the data used in the encryption/decryption does not have a visible effect on the performance of the different algorithms. Finally, the experiments show that one of the most important factors affecting power consumption of an encryption algorithm is the key size.

In this research work, we will consider these results into consideration when it comes to evaluating the performance of the methods developed in this work to enhance the security of database systems while preserving the performance level of the targeted database system.

Sharma *et al*. (2013) presented an approach for achieving data confidentiality, balancing between security and performance by retrieving the data in a quick way. The approach consists of using two tables: Encrypted Data Table which is used to save the desired (usually the primary) column in encrypted form and the Query Search Table which stores the key in encrypted form together with the same column from the Encrypted Data Table, but in plaintext form. The key column is stored under two different names in both tables. This approach removes the confusion happening in range and fuzzy match queries and enhances the performance of query processing. Then, in order to increase confidentiality, the order of the records in the Query Search Table is randomized to add to the confusion in the data. To disguise the relationships, they put the Query Search Table in secure schema and add some noise to the records to prevent inference. Then, only authorized users are allowed to access it. Another advantage is that upon execution of query sentences, it will not need to decrypt all the values in the encrypted column.

Alhanjouri and Al Derawi (2012) proposed the use of Hash Maps to improve the performance of encrypted databases. The authors claim to have devised a method to enhance the response time for queries on encrypted databases. The proposed method involves building an additional layer on top of the DBMS. The layer consists of metadata, a query processor, a hash map and an encryption/decryption function. The authors do not discuss their approach to protect the layer itself, which poses questions on the efficiency of the proposed method in preserving the confidentiality of the data in the first place.

A Reverse Encryption Algorithm (REA) represents a significant improvement over the encrypted databases is proposed by (Mousa *et al*., 2012). The results of REA can reduce the cost time of the encryption/decryption operations and improve the performance, but the encryption of the database is not optimally truthful and it needs some extra security by encrypting the data with another algorithm, to tighten security without degrading performance.

A new approach proposed by (Arasu *et al*., 2014) introduces a system with data encryption where sensitive columns are encrypted before they are stored to address data security.

Zheng-Fei *et al*. (2005; Wang *et al*., 2004) poposed a function to support fuzzy query over the encrypted character data. Their scheme converts every adjacent two characters in the sequence and converts the original string directly to another character string by a hash function. This method cannot deal with some characters and could perform badly for larger character strings.

A novel approach is proposed in this research to improve the performance of the query over encrypted databases based on generating unique hash values for each and every sensitive field and translating the SQL clauses into an appropriate form to execute over an encrypted database. This approach reduces the cost of encryption/decryption and improves query performance.

Architecture Design

Database Management System (DBMS) is a software system that enable users to define, construct, manipulate and share databases between users and applications. Database instances are stored in database; while metadata (data type, structure, constraints, etc.) are stored in data dictionary/database catalog.

DBMS supports set of functions such as: querying the database to retrieve specific data, updating (i.e. insert, update, delete) database. Also DBM provide sharing a database allows multiple users and programs to access the database concurrently.

Users use an application program to access the database by sending queries or requests for data to the DBMS. A query typically causes some data to be retrieved; a transaction may cause some data to be read and some data to be written into the database.

The proposed architecture design is presented in Fig. 1. The clients write his query by using an application program or SQL editor; then the DBMS is responsible to validate and manipulate the user requests to access data. In the propose framework a new layer is used to determine the query has sensitive data or not based on the metadata. If query has sensitive data a new two functions are used; encryption/decryption function and hash map function to retrieve the sensitive data from database.

The proposed approach is divided into two phases. The first phase to store sensitive data by using insert or update statement in SQL. The second phase related to retrieve sensitive data by using select statement in SQL. Each phase is described below:

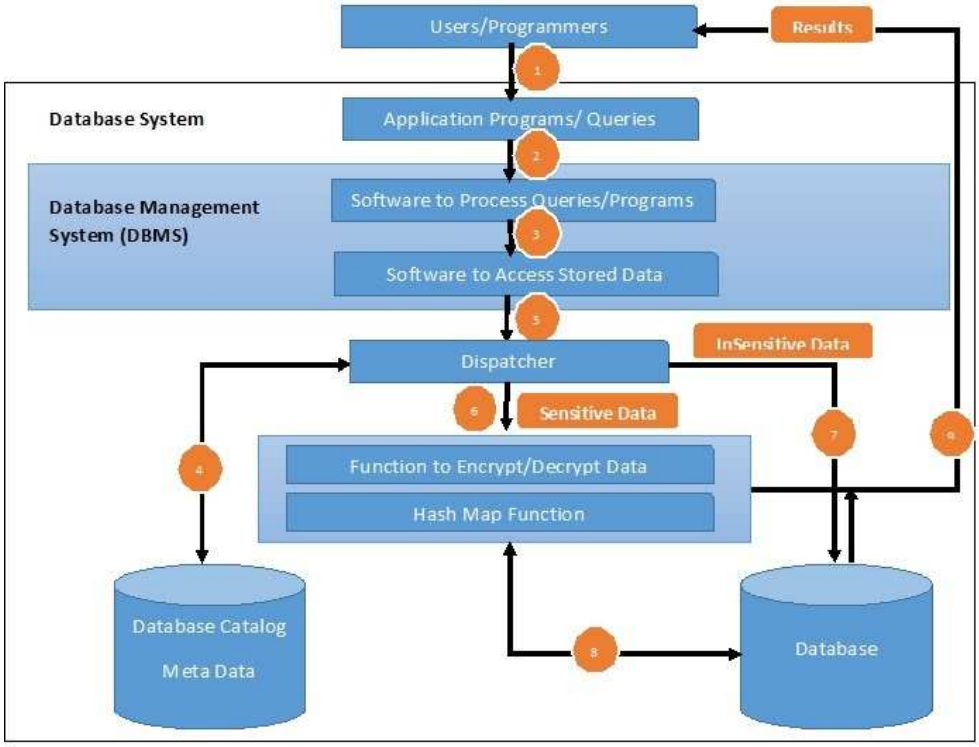

Fig. 1. Database architecture design

Storing Sensitive Data

Storing sensitive data has four steps as following:

Step 1: Generate hash value for sensitive data as following:

- Select two relative prime numbers p, q where $p \neq q$.
- Compute the ASCII code for sensitive field
- Compute hash value H(v) as following:\
  H(v)= [v*Power (p, q)] mod (37*37*37)

Where

H(v) is map hash function
p and q are two relative prime number

Step 2: Encrypt the sensitive data v by using Advance Encryption Standard (AES) with key size 256 bits as following:

c= Enc(v, k)
Where
c: cipher text
Enc: Advance Encryption Standard algorithm
v: sensitive data
k: secrete key with size 256 bits

Step 3: Store the encrypted sensitive data in the original table.

Step 4: Store the hash value and the primary key in the new hash table.

Table 1. Plaintext Customer Table

CustKey	Name	*AccBal*	……...
2935	John	794	……...
4257	Nancy	500	……...
1278	Mike	380	……...
730	Laura	853	……...

Table 2. Encrypted Hash Customer Table

CustKey	Name	*Enc_AccBal*	*Hash_AccBal*	…….
2935	John	Tl?5gfd	272342	…….
4257	Nancy	0(*hf8	171500	…….
1278	Mike	W3b674	130340	…….
730	Laura	xLnbr1	291550	…….

The propose approach uses a new column to store hash map value. When a table is created and contains a sensitive data (s). The DBMS created hash columns include all hash values for sensitive columns. The encryption performed by using AES-256.

For example, as presented in Table 1, let us consider a *Customer* table. The *Customer* table contains eigh fields: *CustKey*, Name, Address, NationKey, Phone, *AccBal*, MktSegment and Comment. The *AccBal* considered as sensitive fields as represented in Table 1.

To provide database security, the sensitive data in the field *AccBal* is encrypted by using AES with key size 256 bits. As shown in Table 2. The field *Enc_AccBal* represent encrypted *AccBal* value and *Hash_AccBal* represented the hash value for *AccBal* value by using hash map function.

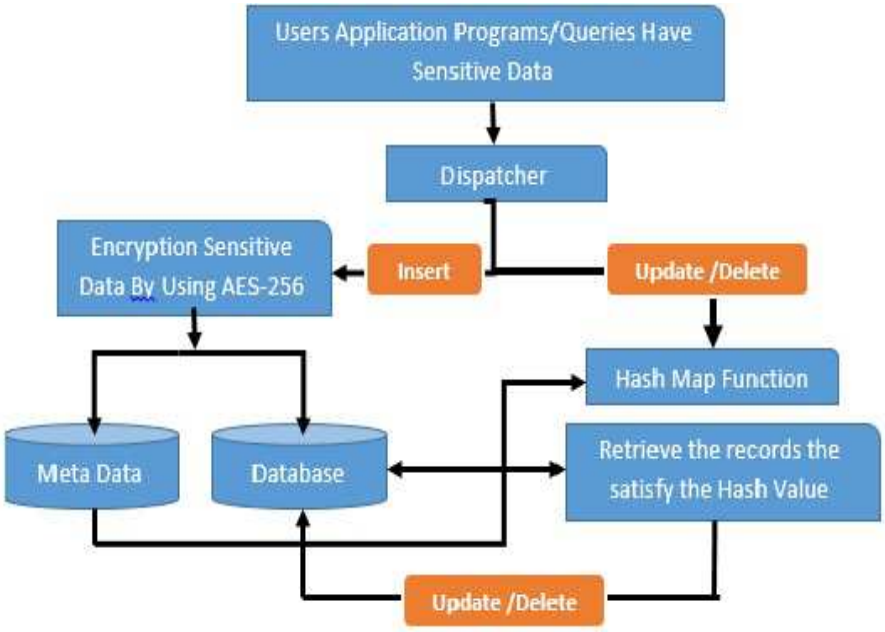

Fig. 2. Architecture design for storing sensitive data

The Table 1 is extended by adding two fields one for encrypted *AccBal* and the other for hash value. Indexes are created for the fields *Enc_AccBal* and *Hash_AccBal*. Figure 2 illustrates the process of storing sensitive data into encrypted database.

Query Over Encrypted Database

When where clause in select statement has encrypted field the following steps are executing:

Step 1: Compute hash value for input data as following:
Retrieve the prime numbers p, q which stored in database in secure manner.

- Compute the ASCII code for input data
- Compute hash value H(v) as following:
- H(v)= [v*Power (p, q)] mod (37*37*37)

Step 2: Translate and modify the query conditions of SQL by using the metadata.
Step 3: Execute the modified SQL query
Step 4: Retrieve all records that satisfy the hash value of query condition which computed in step 1
Step 5: Decrypt all records that retrieved from step 4.
Step 6: Return results.

Figure 4 illustrates the SQL query over encrypted database. For example, if you have the following SQL query:

Select *CustKey, Name, AccBal*
From *Customer*
Where *AccBal* >= 563;

The proposed approach translates the above SQL query as following:

Select *CustKey*, Name, Decrypt(*AccBal*)
From *Customer*
Where *Hash_AccBal* >= Hash(563);

Experiments

In this section, a set of experiments were carried out to validate the performance and effectiveness impact of the proposed approach.

Experiments Environment

The experiments were carried out on a server intel Xeon 5600 series with 2 processor 3.46 GHz, cache 8 MB L3, 192 GB RAM. The operating system that was used is Microsoft Windows Server 2012 R2. Oracle 12c R2 was used as a DBMS. The programming task in the proposed approach was implemented by Java programming language. Each experiment was executed 5 times and the average of results were considered. The database was generated automatically by using Dbgen tool according to TCP-H benchmark (TPC-H Benchmark Specification, 2017). TCP-H database contains eight tables. *Customer* table was used in our experiment. The *AccBal* was considered as sensitive field. The column of *AccBal* was encrypted by using AES-256 to provide confidentiality service.

The proposed approach and set of related approaches from literature were examined; to quantity the CPU execution time of SQL operations over the encrypted column in *customer* table, which has a number of tuples ranging from 50,000 to 100,000. The following queries were carried out in those experiments:

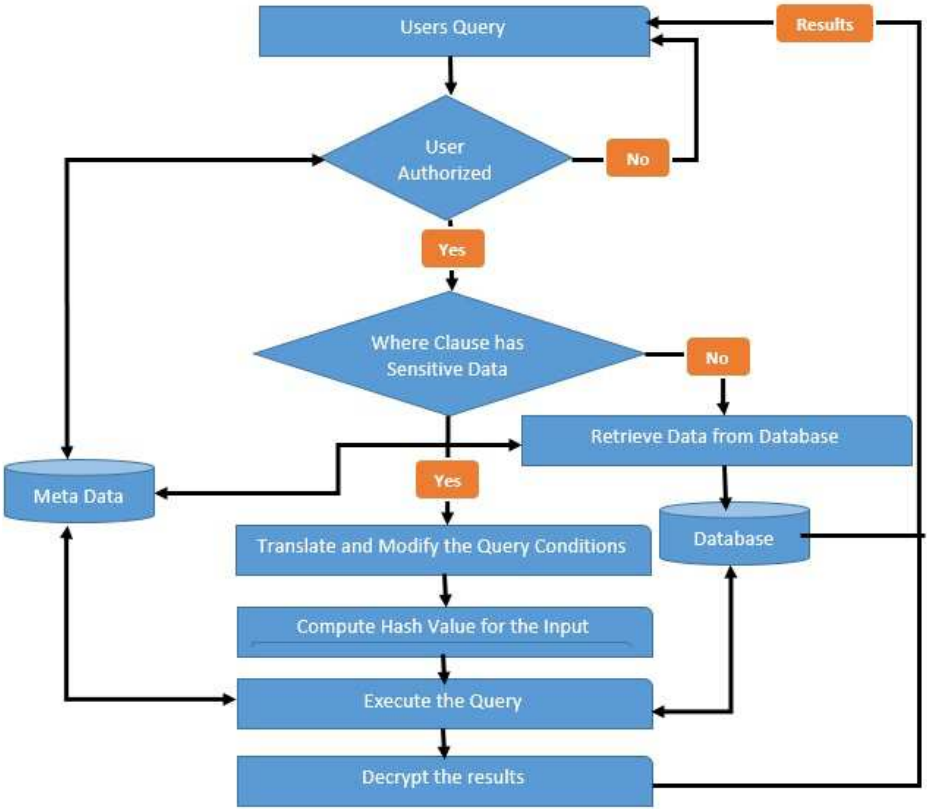

Fig. 3. Architecture design for SQL query over encrypted database

Query 1: Select Statement

The where clause in the query statement has an encrypted field.

Select *CustKey, Name*
From *Customer*
Where *AccBal between 2564 and 8729*
Based on our approach the above query was transformed to:
Select *CustKey, Name, Dccrypt(AccBal)*
From *Customer*
Where *Hash_AccBal between Hash(2564) and Hash(8729)*

Query 2: Select Statement

The query statement has an encrypted field and where clause has encrypted field.

Select *CustKey, Name, AccBal*
From *Customer*
Where *AccBal between 2564 and 8729*
Based on our approach the above query was transformed to:
Select *CustKey, Name, Dccrypt(AccBal)*
From *CustKey, Name, Dccrypt(AccBal)*
Where *Hash_AccBal between Hash(2564) and Hash(8729)*

Query 3: Update Statement

The where clause in update statement has an encrypted field.

Update *Customer*
Set *Phone = Phone + 5*
Where *AccBal between 2564 and 8729*
According to the propose approach the update statement was transformed to:
Update *Customer*
Set *Phone = Phone + 5*
Where *Hash_AccBal between Hash(2564) and Hash(8729)*

Query 4: Update Statement

The update statement and where clause have an encrypted field.

Update *Customer*
Set *AccBal = AccBal + 730*
Where *AccBal between 2564 and 8729*
The above query was transformed to:
Update *Customer*
Set Decrypt(AccBal) = Encrypt(AccBal) + 730
Hash_AccBal = Hash(Encrypt(AccBal)+730)
Where *Hash_AccBal between Hash(2564) and Hash(8729)*

Query 5: Delete Statement

The where clause has encrypted field in delete statement.
Delete *Customer*
Where *AccBal between 2564 and 8729*
This query was transformed to:
Delete *Customer*
Where *Hash_AccBal* between Hash(2564) and Hash(8729)

Query 6: Insert Statement

The insertion execution time was computed to determine the time that was consumed by insertion operation with encryption process.

Insert into *Customer*
values *(CustKey, Name, Encrypted(AccBal),….);*

Evaluation Metric

In this research, the CPU execution time (second) was considered. The evaluation performance encryption/ decryption process and hash map function process conducted in term of the CPU execution time of select, update, delete and insert statement. In this research work the all above queries were executed on the *Customer* table by using the following approaches:

1. Traditional approach: which decrypt all encrypted data before applying the SQL query.
2. Sharma *et al*.'s method.
3. Alhanjouri and Al Derawi approach.
4. Wang *et al*.'s approach
5. Proposed approach.

To obtain fair and fixed comparison between all above approaches, same data and same functions that are responsible for accessing the database were used in all experiments. Each experiment was tested with following data size 50,000 and 100,000 records.

Results Analysis and Discussion

The results of our experiments in term of CPU execution time or query was presented in following sub sections.

Execution Time for Query 1: Select Statement

The select statement in this query does not contain any encrypted field; while where condition has encrypted field. As showed in Fig. 4 and 5, the average of CPU execution time 18s, when select statement was executed over *customer* table with datasize 50,000 records. While the CPU execution time was 32s when the same query executed in data size 100,000 records.

There is a significant improvement in execution query over the encrypted database compared with other approaches, as shown in Fig. 4 and 5. The CPU execution time was 30s, 26s and 23s for traditional approach, Sharma *et al*. and Alhanjouri and Al Derawi methods; respectively when data size is 50,000 records. While the CPU execution time was 58s, 47s and 42s when data size is increased to 10,000 records.

Execution Time for Query 2: Select Statement

In this scenario select statement and where condition, both of them contain encrypted field. Therefore; the execution time will be increased compared with previous scenario in Query 1.

When Query 2 was executed over *customer* table with data size 50,000 and 100,000 records, the response time was 40s, 62s; respectively in the proposed approach. The response time is increased, when other approaches were executed over encrypted database as presented in Fig. 4 and 5.

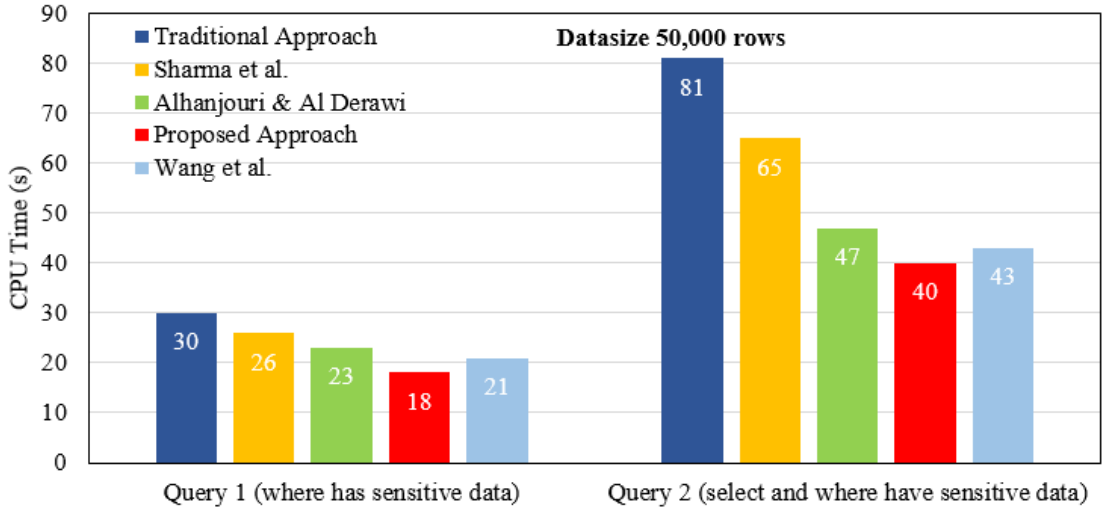

Fig. 4. Execution time for select statement (Query 1 and 2)

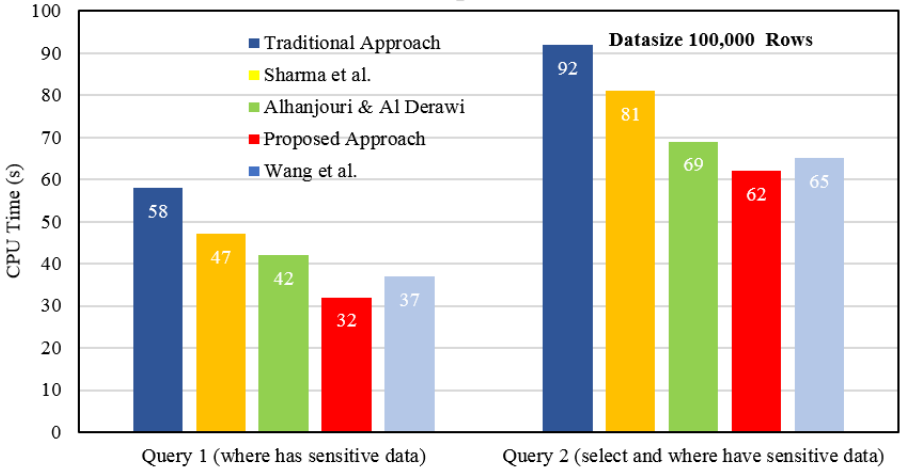

Fig. 5. Execution time for select statement (Query 1 and 2)

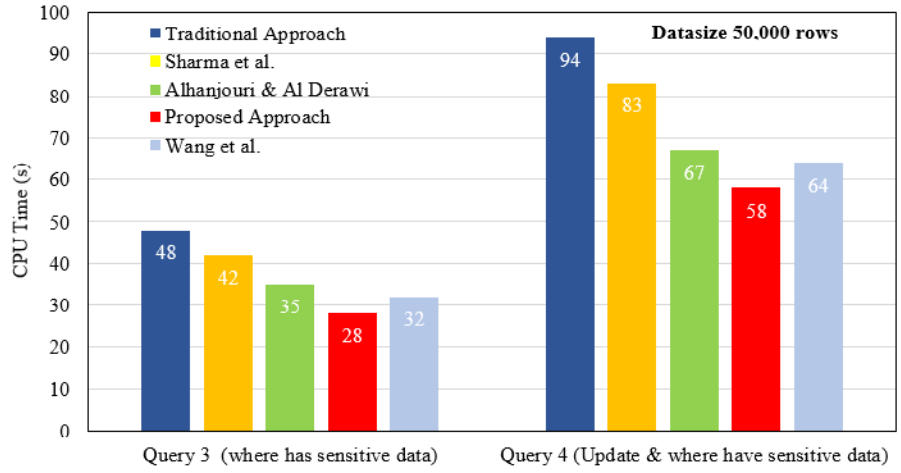

Fig. 6. Execution time for Update statement (Query 3 and 4)

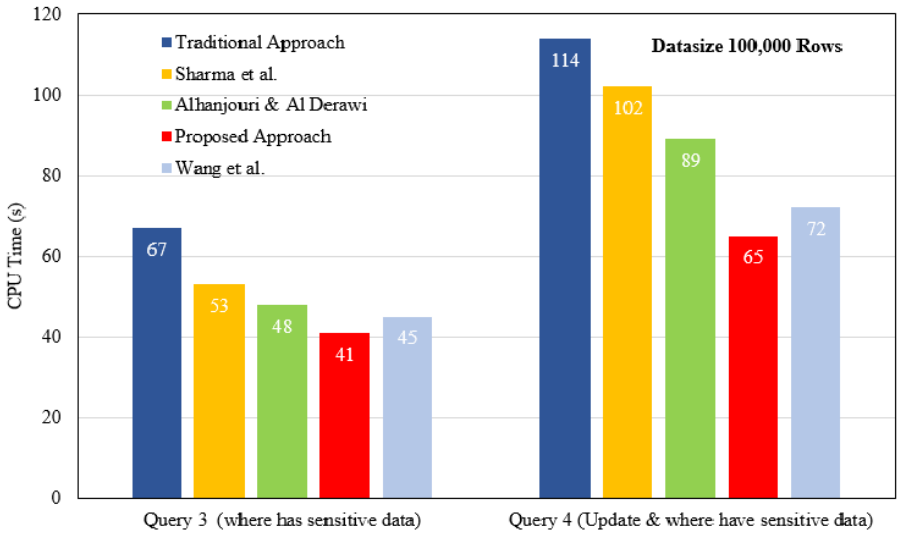

Fig. 7. Execution time for Update statement (Query 3 and 4)

Execution Time for Query 3: Update Statement

Update statement is not containing any sensitive data, while where statement has sensitive data. Therefore; the proposed approach translates the query to compute the hash value for sensitive data. In this case the average of execution time for this query was less than 28s and 41s when database size is 50,000 and 100,000 records; respectively. The results of this experiment was presented in Fig. 6 and 7.

Execution Time for Query 4: Update Statement

Both update statement and where condition have encrypted data. Therefore; the proposed approach translates the query to compute the hash value for sensitive data to retrieve the records that satisfy the where condition, then decrypt the sensitive value and then encrypt the results. In this case the average of execution time for this type of query greater than the previous query 3 because there were two operations are performed. The CPU execution time was 58s for database size 50,000 rows; while for 100,000 records, the execution time was 65s. The results of experiment illustrated in Fig. 6 and 7.

Execution Time for Query 5: Delete Statement

In delete SQL statement, only where condition contains encrypted data. Therefore; the proposed approach calculated hash value, then find all rows satisfy the condition. For this scenario, the response time was 28s for database size 50,000 rows; while for 100,000 records, the response time is 54s. The results of this scenario presented in Fig. 8 and 9.

Execution Time for Query 6: Insert Statement

Figure 10 shows the relationship between encryption time during insertion the row in the encrypted database and the number of rows for each approaches. When the size of data increases, the CPU execution time also increase. From experiment results, it is obvious that the proposed approach consumes the longest time for encrypting and hashing; while the traditional approach consumes least time for that; because the traditional method perform only encryption, no need for hashing the sensitive data. The proposed approach need extra time for hashing sensitive data, but has better execution time than Sharma *et al*. for insertion operation.

The results of all experiments validate the performance of the proposed approach. There is a significant improvement in execution query over the encrypted database. This performance improvement in execution query over encrypted database and minimized CPU time cost; because the proposed approach being based on computing a hash value for where clause conditions, then selecting all records that satisfy the hash value for where conditions. While other approaches based on decrypt all records in the *customer* table then retrieve the records that satisfy the where clause conditions. Also, Sharam *et al*. used two tables for a single main table. The first table contains the actual data (*CustKey*, Name, Encrypt (*AccBal*),….), which has its sensitive data in encrypted form, while the second table contains Encrypt (*CustKey*), *AccBal*. This method consumes CPU time when executing queries over the encrypted table.

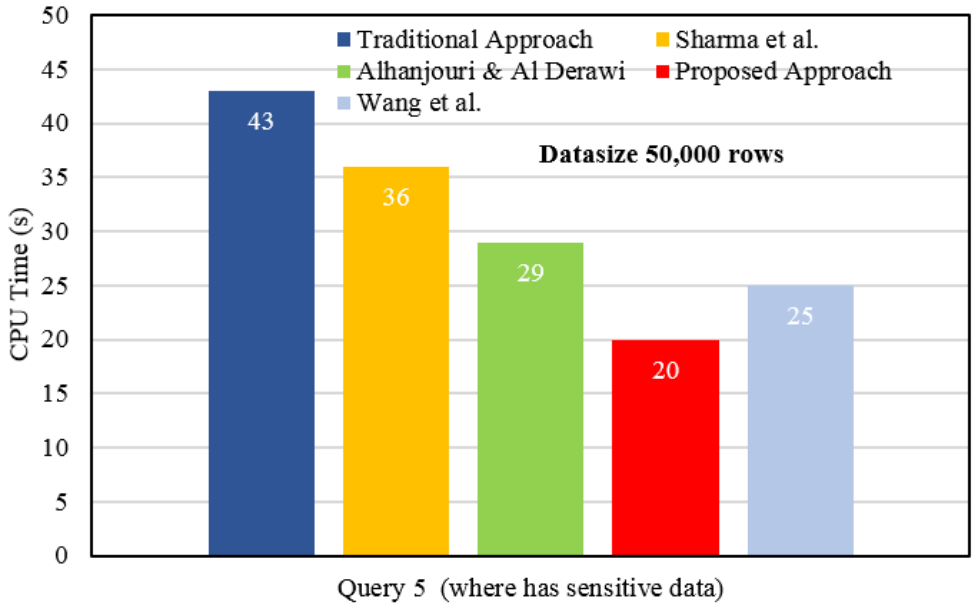

Fig. 8. Execution time for delete statement (Query 5)

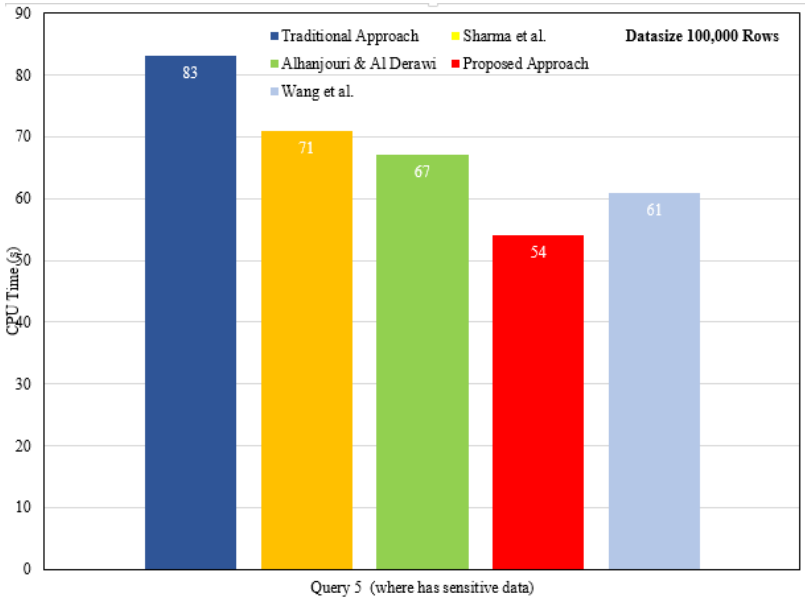

Fig. 9. Execution time for delete statement (Query 5)

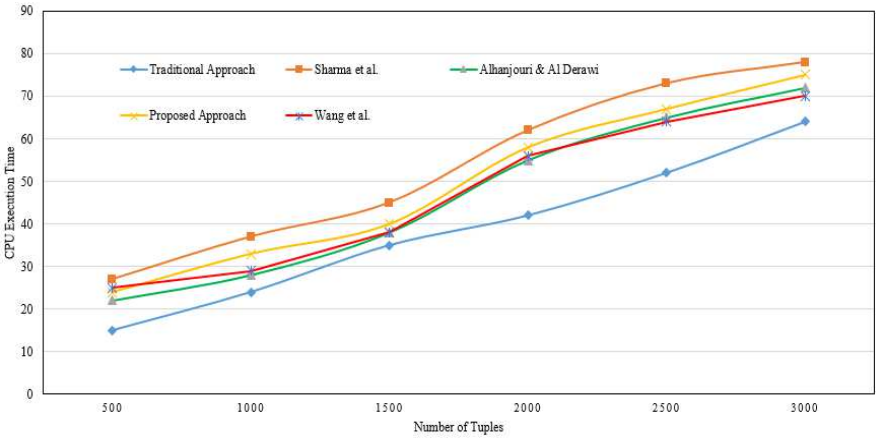

Fig. 10. Execution time for Insert statement (Query 6)

Conclusion

This research has presented an enhancement to naïve database encryption approach to achieve better response time for the execution of SQL query. The experimental results indicate that the adaptive approach improves the performance of query response time for all cases, providing excellent query response time and improve query performance using a variety of records. The outcomes demonstrate that the proposed methodology gives better response time and performance.

Ethics

This article is the original contribution of the author and is not published elsewhere. There is no ethical issue involved in this article

References

Alhanjouri, M. and A.M. Al Derawi, 2012. A New method of query over encrypted data in database using hash map. Int. J. Comp. Appl., 41: 975-888. DOI: 10.5120/5533-7580

Ali, A. and M.M. Afzal, 2017. Database security: Threats and solutions. Int. J. Eng. Inventions, 6: 2278-7461.

Arasu, A., K. Eguro, R. Kaushik and R. Ramamurthy, 2014. Querying encrypted data. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 22-27, ACM Press, New York, USA, pp: 1559-1261. DOI: 10.1145/2588555.2588893

Bouganim, L. and Y. Guo, 2011. Database encryption. Proceedings of the Encyclopedia of Cryptography and Security, Boston, MA: Springer US, pp: 307-12. DOI: 10.1007/978-1-4419-5906-5_677

Mousa, A., E. Nigm, S. El-Rabaie and O. Faragallah, 2012. Query processing performance on encrypted databases by using the REA algorithm. Int. J. Network Security, 14: 280-88.

Nassar, M., Q. Malluhi, M. Atallah and A. Shikfa, 2017. Securing aggregate queries for DNA databases. IEEE Trans. Cloud Comput. DOI: 10.1109/TCC.2017.2682860

Rathod, R.H. and C.A. Dhote. 2014. A literature survey on performance evaluation of query processing on encrypted database. Int. J. Eng. Comput. Sci., 3: 9637-9642.

Salama, D., A. Elminaam, H. Mohamed, A. Kader and M.M. Hadhoud, 2010. Evaluating the performance of symmetric encryption algorithms. Int. J. Network Security, 10: 213-19.

Sharma, M., A. Chaudhary and S. Kumar, 2013. Query processing performance and searching over encrypted data by using an efficient algorithm. Int. J. Comput. Appli., 62: 975-8887. DOI: 10.5120/10114-4781

TPC-H Benchmark Specification, 2017. http://www.tpc.org/tpch/

Wang, Z.F., J. Dai, W. Wang and B.L. Shi, 2004 Fast query over encrypted character data in database. Communications Information Syst., 4: 289-300.

Zheng-Fei, W., W. Wang and B.L. Shi, 2005. Storage and query over encrypted character and numerical data in database. Proceedings of the 5th International Conference on Computer and Information Technology, Sept. 21-23, IEEE Xplore press, Shanghai, China, pp: 77-81. DOI: 10.1109/CIT.2005.174