Searchable Encrypted String for Query Support on Different Encrypted Data Types

Shahrzad Azizi[1] and Davud Mohammadpur[1*]
[1] University of Zanajn
Zanjan, Iran
[e-mail: azizi.shahrzad@znu.ac.ir, dmp@znu.ac.ir]
* Corresponding Author: Davud Mohammadpur

Received March 29, 2020; revised June 3, 2020; revised July 12, 2020; revised July 22, 2020; accepted September 5, 2020; published October 31, 2020

Abstract

Data encryption, particularly application-level data encryption, is a common solution to protect data confidentiality and deal with security threats. Application-level encryption is a process in which data is encrypted before being sent to the database. However, cryptography transforms data and makes the query difficult to execute. Various studies have been carried out to find ways in order to implement a searchable encrypted database. In the current paper, we provide a new encrypting method and querying on encrypted data (ZSDB) for different data types. It is worth mentioning that the proposed method is based on secret sharing. ZSDB provides data confidentiality by dividing sensitive data into two parts and using the additional server as Dictionary Server. In addition, it supports required operations on various types of data, especially LIKE operator functioning on string data type. ZSDB dedicates the largest volume of execution tasks on queries to the server. Therefore, the data owner only needs to encrypt and decrypt data.

Keywords: Data Encryption, Queryable Encryption, Secure Databases, Secure SQL Queries, Secret Sharing

http://doi.org/10.3837/tiis.2020.10.015 **ISSN : 1976-7277**

1. Introduction

There are numerous threats against information privacy and security in data storing, especially data outsourcing. These threats can be categorized as internal or external. Due to the sensitive nature of data, maintaining confidentiality is the challenge that appears. Confidentiality of data makes data content unavailable to unauthorized users. In this way, data owners need to encrypt data before storing sensitive data and executing queries [1], [2].

Data encryption [3], particularly application-level encryption, is a common solution to protect data confidentiality and face security threats. Application-level encryption is a process in which data is encrypted before being sent to the database [4]. Application level Data Encryption has several benefits including,

- Safe against alteration: Others cannot change data without permission.
- Ensure compliance: One can be sure that rules and policies on data are followed [4].
- Backups are safe: Others cannot use backup.
- Prevent data leakage: Data capture will not be possible through eavesdropping [5].
- Move data Securly: The transfer and use of data will be in secure condition.

In order to reach the abovementioned benefits, data owner converts his/her data into an encrypted form before storing. Since encryption and data transformation can lead to the elimination of the main features (for example, length and format) of data, an important challenge that arises is the execution of SQL queries on encrypted data, which will be addressed in this study.

Homogeneous encryption methods such as FHE as well as PHE can be used to execute queries on the encrypted data. The purpose of homomorphic encryption is to allow computation on encrypted data. The FHE encryption methods make it possible to perform all types of processing and execute all types of queries on encrypted data. However, the important issue is the high cost of execution. Therefore, these methods are used limitedly and have had no widespread acceptance so far [6]. In contrast, PHE methods allow only some types of processing and queries on the encrypted data. Since the type of data is specific for each PHE methods, it is necessary to use a different method for each data type. The use of these methods is complicated and infeasible [7]. Another method is the secret sharing [8]. In secret sharing, each sensitive data element, called a secret, is split into n shares, which are distributed to multi server, and no one can recover the plain values by its own shares [9].

Although the purpose of the mentioned methods is to provide a suitable solution to the problem of executing queries on encrypted data, an appropriate one has not been provided yet. Without a suitable solution, databases with encrypted data cannot be used to protect data, and users have to store data without encryption which can bring about various data threats.

The current study aims to present a secure query executing method based on the secret sharing model. Since the server is considered as trusted but curious one, the server stores the sensitive data in a encrypted form, and the keys are retained by the data owner. Accordingly, data cannot be decrypted on the server and data security is well guaranteed.

The rest of the paper is organized as follows: Section 2 presents the related works. Section 3 discusses the proposed architecture, the data model and the main idea. Section 4 is about the proposed method concerning how to encrypt and decrypt the data. Security analysis is

provided in Section 5. Section 6 represents evaluation of the proposed method. Finally, Section 7 concludes the paper.

2. Related Works

Review of the related works shows that there have been countless attempts to develop queryable encrypted databases. These attempts should be able to support required operations on various data types in addition to maintaining the confidentiality of data. Since the operators on each data type are different, encryption methods should have various facilities. Owing to the importance of supporting all data types (numerical and string), related methods are examined based on their capability to support various data types. Most methods support numerical data types and a limited number of them are provided for string data.

2.1 Numerical Data Type

Generally, in order to provide information security and prevent internal attacks, three main approaches have been developed: Index based methods [10], [11], Homomorphism encryptions [12], [13] and Secret sharing methods [9], [14]. All of these approaches can be applied to numerical data types.

2.1.1 Index Based Methods

A common technique to speed up the execution of queries is to use pre-computed indexes. However, once the data is encrypted, the use of standard indexes is not possible. In the index base methods to create encryption index, the attribute domain is usually divided into a set of non-overlapping parts. Assigning explicit tags to each part, the attribute values are mapped to the corresponding part. In encrypted databases, the encryption index has a significant role in query performance and can accelerate it [10].

An early work [10] suggests encrypting the whole record and assigning a set_identifier to each value in the record. When searching a specific value, its set_identifier is calculated and then passed to the server. The server returns a collection of all records with values assigned to the same set to the client. Finally, the client searches the specific value in the returned collection and retrieves the desired records. This method is suitable for executing equation conditions and range queries, but it is not possible to perform aggregation functions such as SUM, MIN and MAX [15]. To meet the requirements of aggregation functions, in [11], authors made an attempt to use preprocessing functions and additional tables which leads to high cost and extra overhead.

2.1.2 Homomorphic Encryption Based Methods

Various attempts such as CryptDB [12] have been made regarding homomorphic encryption. CryptDB has proposed the idea of encrypt attributes at different levels, such as onion layers [16]. It also uses a reliable proxy server to store encryption keys, database schema and onion layers of all attributes. Moreover the data encryption, rewriting queries and decryption of results are the responsibility of the proxy [17]. In order to process the query, the proxy checks the required attributes and then considers the suitable layer. Consequently, it separates the onion layers dynamically, if necessary, and assigns data computation to the appropriate layer [16]. However, it is significant to note that the inner layers do not provide high level of security and are vulnerable to attacks. Furthermore, onion layers create overhead, especially in

the case of large tables. It should be stated that CryptDB cannot support string LIKE queries and analytic queries [17].

MONOMI [13], which is based on CryptDB, is capable of supporting analytical queries. It uses several optimization techniques to realize this goal. It also uses a designer to optimize the data plan [18].

2.1.3 Secret Sharing Methods

In cryptography, secret sharing refers to any method for distributing a secret among a group of participants, each of which allocates a share of the secret. The secret can only be reconstructed when the shares are combined together. Individual shares are of no use on their own [19]. Secure Query Processing System (SDB) method [9] uses the secret sharing to encrypt data. As for SDB, each sensitive data is divided into two parts in order to process a secure query[20]. A portion of data is stored on the data owner's side that is trustful, and the other part is stored on the server's side which is unreliable. Non-sensitive data is stored in plain text on the server's side. SDB also provides various operators that can be applied to encrypted data, plain text data, or a combination of them [8]. However, SDB is only designed to support numerical data types and cannot support string types [21].

2.2 String Data Type

Methods that implement queryable encrypted databases on string data types need to support LIKE operator to search on encrypted data. These methods can be considered in three categories: based on Bloom filter [22], [23], based on mapping [24], [25] and converting string to number [26], [27].

2.2.1. Based on Bloom Filter

These methods convert each word into a vector using the Bloom filter and its hash functions. Then, in the query execution, they convert entered string to a similar form. Finally, by using the Euclidean distance calculation, the similarity of the entered string is determined with the stored values. One of the weaknesses of these methods is the possibility of false positive results. In addition, these methods are only suitable for searching and exact matching but cannot support wildcards well.

2.2.2. Converting string to number

Some methods initially convert strings to numbers and then store them on the server. These methods use the ASCII code and Unicode to encrypt English and Persian letters, respectively. Then, they store their numerical equivalents on the server after applying encryption methods. These methods are only suitable for searching and exact matching but cannot support wildcards well.

2.2.3. Based On Mapping

Some methods have used mapping to support string data. SQL-based fuzzy query mechanism over encrypted database (FQE) method in [24] initially extracts all words from the text and then produces all combination of its unigram, bigram, and trigram forms for each word as its statements. After that, it maps each statement to a Unicode character. Finally, using a random algorithm shuffles the characters and saves the result. The obtained result is used in querying steps, especially in LIKE operator. In addition, in order to decrypt, FQE adds another

additional column to the table whose values are obtained using a cryptographic function. FQE uses this additional column to decode the results.

3. Proposed Architecture

This paper provides ZSDB method that supports efficient data encryption and querying on numeric and string data types simultaneously. As for ZSDB, data owner only needs to encrypt data and decrypt results. Accordingly, the most workload is transferred to the server. This method uses an additional server for storing some metadata to support the string data.

As shown in **Fig. 1**, the proposed architecture has three main components, including data owner (*Client*), dictionary server (*Server1*) and encrypted data server (*Server2*). Initially, data owner (*Client*) sends some encrypted words to the dictionary (*Server1*). *Server1* sets an index for each one. In fact, *Server1* stores a subset of words in order to support the string queries. Then, data owner encrypts these indexes in a way similar to the method provided for numerical data and stores them along with other encrypted data on *Server2*.

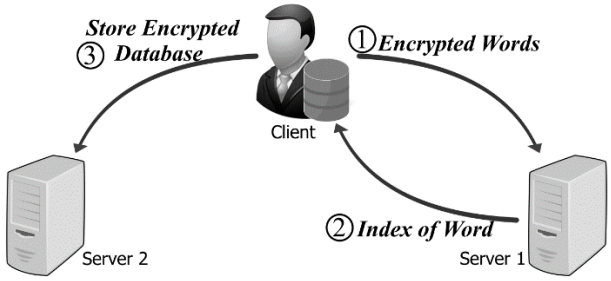


Fig. 1. Storing Data in Proposed Architecture

When the user sends a query to the data owner (*Client*), if the query does not contain LIKE operator, *Client* translates the query and then requests it from *Server2*. Finally, after decrypting the returned results from *Server2*, they are presented to the user in plaintext.

If the request contains LIKE operator (**Fig. 2**), data owner (*Client)* must first request indexes of the word from *Server1* in order to verify the word availability on the server. Then, when indexes are available, **Fig. 2**, it translates the corresponding query and requests from *Server2*. *Server2* sends the encrypted results to *Client*. After decryption, *Client* sends the plaintext result to the user.

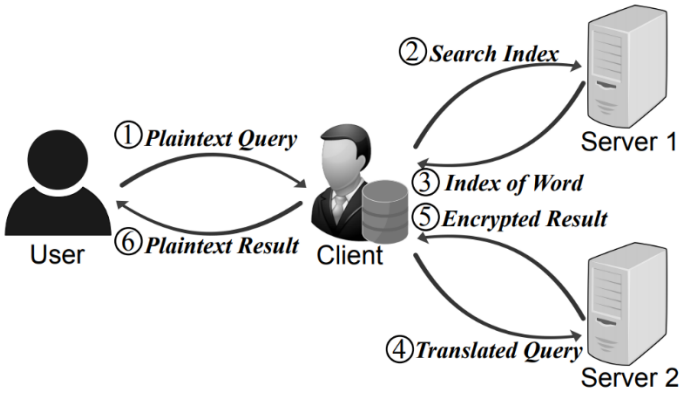


Fig. 2. Querying LIKE operator in the Proposed Architecture

4. Proposed Method

The purpose of ZSDB is to provide secure storage and query execution on a variety of data types, especially string data type. Accordingly, the proposed method includes some steps to create data dictionary, encryption, querying and decryption. In what follows, subsections describe details of each one for each data type. How to implement the proposed method is different for each type of data, so in the following section, how to proceed with each type of data is presented separately.

4.1. Integer Data Type

4.1.1 Encryption

The proposed method is based on the SDB [9], and integer data type values encryption is same as SDB method. In this case, numbers n and g are considered and stored on the data owner's side. n is obtained from the product of two prime random numbers p and q, and g is a positive number that is co-prime with n.

In this method, we consider a pair of random numbers m and x for each column of sensitive data (c_l) as the column key c_l <m,x> whereas m, $x < n$ and assign a random positive value r_k to each row as the row number. r_k can be encrypted by homogeneous methods and stored in the final table (*Server2*) along with other record information (for example, SIES [28]).

According to the secret sharing method, each data is divided into several parts. We divide it into two parts: key and encrypted value in which the value of the key and the encrypted value are stored on the client and server's side, respectively.

Key generation: At this stage, considering the number of the row r_k and c_l <m,x> for the column, the key of any sensitive data v_k, is calculated according to (1) [9]. It should be noted that there is no need to store v_k values on the data owner's side. It can be generated based on g, r_k and c_l <m,x> values.

$$v_k = gen(r_k, < m, x >) = mg^{r_k x} \bmod n \tag{1}$$

Producing encrypted value: After key generation, (2) obtains the encrypted value for a sensitive value ($||v||$).

$$v_e = \varepsilon(\|v\|, v_k^{-1}) = \|v\| v_k^{-1} \bmod n \tag{2}$$

This value is calculated by the multiplication of inverse values of the key and sensitive data, in which, (3) obtains the inverse value [8].

$$v_k v_k^{-1} \bmod n = 1 \tag{3}$$

Based on this method, multiplication, addition, subtraction, comparisons and other operations can be supported well, which can be studied further in [9].

4.1.2 Decryption

To retrieve stored data, first, data owner requests the encrypted (v_e) value, which is stored on the server. Then, according to (4), multiplies v_e by the value of the key v_k that can be calculated on its side based on g, r_k and c_l <m,x> values. Finally, the remainder of the number n is equal to the value of the sensitive data [9].

$$\|v\| = v_e v_k \bmod n \tag{4}$$

Fig. 3 shows the encryption and decryption steps in the proposed method in which the value of the column key, *n* and *g* numbers are stored on the data owner's side (The value of r_k is equal to one for all records).The encrypted values are stored on the server's side.

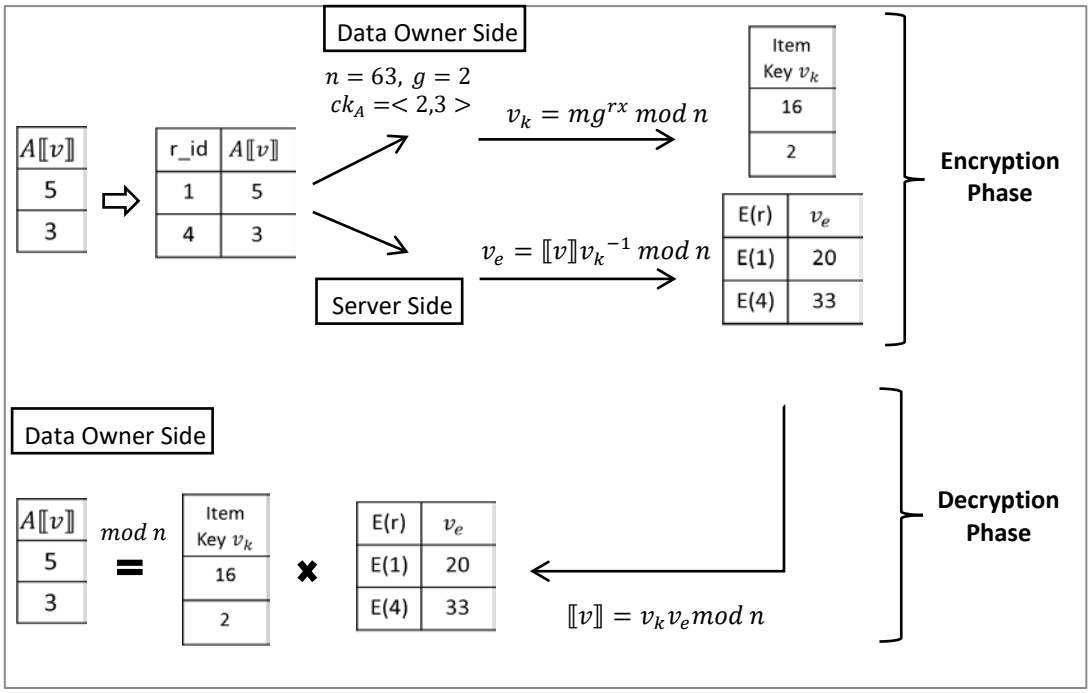


Fig. 3. Integer Data Type Encryption and Decryption Steps

4.2 Real Data Type

In order to encrypt real data type values, we consider values (*N*) in the form (5). Then, assigning proper values to *s* and *M*, we convert the value of *N* to *M* and *s*.

$$N = M \times 10^{S} \tag{5}$$

According to (5), the result values (*M* and *s*) are integer type that can be encrypted as the proposed method for the integer data type values.

4.3 String Data Type

The proposed method includes some steps for string data type values, including creating data dictionary of substrings, encrypting based on substrings indexes, querying based on substrings relationships and decrypting the results.

4.3.1 Creating Data Dictionary

In order to support string values, we use a dictionary to store a sub-set of words and put it on *Server1*. First of all, data owner (*Client*) extracts unigram, bigram, and trigram of each word of the string and stores them in the WORD column of *DICTIONARY* table on *Server1*. For example, **Table 1** shows a part of *DICTIONARY* table for "davud" and "data" words.

In the queries related to the string values, especially to support LIKE operator, we consider three types of matching: substring matching (%str%), prefix matching (str%) and suffix matching (%str). To support them, in *DICTIONARY* table, phrases are organized in the form of

a tree in which each unigram refers to its associated bigrams as its *Childs*, and each bigram refers to its associated trigrams as its *Childs*. It helps search for substrings. For instance, in **Table 1**, *WORD* column contains encrypted phrase and the *childL*, *childLL*, *childR* and *childRR* refer to left child (suffix matching), left-left child (suffix matching), right child (prefix matching), right-right child (prefix matching). Left child of a word is all words that begin with that word. Right child of a word is all words that end with.

Creating *DICTIONARY* table is a dynamic action, and data owner updates the table for each new word. In fact, data owner at the same time as updating the table on *Server1*, encrypt the indexes and store encrypted data in *Server2*.

Table 1. Example of the DICTIONARY table

Index	WORD	ChildL	ChildR	ChildLL	ChildRR
00000001	d	4	13	7,14	16
00000002	a	5,11	4, 6	8,15	8
00000003	t	6	5		7
00000004	da	7,14			
00000005	at	8	7		
00000006	ta		8		
00000007	dat				
00000008	ata				
00000009	v	12	11	16	14
00000010	u	13	12		15
00000011	av	15	14		
00000012	vu	16	15		
00000013	ud		16		
00000014	dav				
00000015	avu				
00000016	vud				

4.3.2 Encryption

For storing string data, depending on the length of the word, data owner sends the unigram, bigram, or trigrams to *Server1* and receives indexes of the word from *Server1*. Words with a length greater than three characters is mapped into trigrams. Finally, data owner stores encrypted form of indexes in *Server2*. For example, for "data", data owner considers the index of both "dat", "ata" indexes. Since the index column is 8-byte, the index is encrypted as two 4 bytes. Therefore, the word "data" is ciphered and stored as E(0)E(7)E(0)E(8).

4.3.3 Searching

Concerning searching, if the user requests {LIKE "dat%"}, initially data owner extracts the index of "dat" from *Server1*, and after translating the query, requests {LIKE "E(0)E(7)%"} from *Server2*. In the next step, when *Server2* returns the results associated with the search term, the results are the indexes of the words. Therefore, data owner sends the indexes to *Server1*. *Server1* returns the statement of each index and data owner concatenates the statements and gives the result to the user in plain text. The following subsection discusses details of decryption.

4.3.4 Decryption

In the decryption of the string values, when *Server2* returns results, each result is broken into words. Then, for each word, data owner converts a pair of characters to a number. In the next step, after summing up, for each 8-byte number, data owner requests the term associated with it from *Server1*. Concatenating the results, data owner generates data in plain text and gives them to the user.

5. Security Properties of ZSDB

In this section, an overview of security properties of ZSDB in encrypting, querying, and decrypting results is provided.

Regarding security, there are two categories of attackers that can break data confidentiality, privacy as well as integrity in outsourced databases, namely outside and inside attackers [1]. Facing these two categories of attackers, following security requirements should be taken into account [1], [29]:

1. Encrypted data: An untrusted server should not get any information about the original data through the results.
2. Query restriction: Only data owner and authorized users can query on stored data and server cannot do that.
3. Encrypted queries: The queries must not reveal any information about the data on the server's side.
4. Decrypting in trusted side: The results must be available only for the owner so that others cannot access plain results.

The following is an evaluation of ZSDB method in terms of above requirements.

5.1 Encrypted Results

In ZSDB, *Server2* only observes some encrypted numbers obtained from some of the indexes in *Server1*. In this case, since the form of the words contained in *Server1* is in form of unigram, bigram and trigram, even if the two servers collaborate, there is no way to disclose information. Another type of attack is the frequency analysis attack. In ZSDB, the least abundance occurs in the dictionary table. Since we store the highest level of the subset of words in the dictionary table, this type of attack will be failed.

5.2 Query Restriction

In order to search in ZSDB, unigram, bigram and trigram of the words are stored in *Server1* and only sub-set of words can be selected based on the length of the words. Since the words stored in *Server1* are sub-set of words, outside and inside attackers will have no way to search the stored data directly.

5.3 Encrypted Queries

In ZSDB, servers can search and query on encrypted data and do not need to get plain queries. Clients translate queries based on *Server1* indexes and send only encrypted queries to *Server2*, so no server will receive a plain query. *Server2* runs the received query as a completely normal SQL query. Therefore, it will not be aware of the query encryption at all. *Server2* returns the results in the form of encrypted data.

5.4 Decrypting in Trusted Side

After receiving the encrypted results from *Server2*, data owner decrypts the values based on *Server1* indexes, and concatenates the values returned from *Server1* to forms plain data. In this way, plain data will only be available on data owner's side, and the servers will not have any access to the decrypted data.

6. Evaluation of ZSDB

ZSDB encrypts numerical data types based on SDB method. In [9], SDB has been evaluated by other methods, and the ability to support a variety of operators and the reliability of the results have been confirmed in a number of papers such as [19], [21], and [26]. Therefore, in this section, we only evaluate ZSDB in terms of the string data type. We analyze the results from a storage overhead and execution time perspective. Among the methods used to support the type of string data type, we chose and implemented FQE method [24]. It is worth noting that the chosen method is not based on the SDB method. However, according to the reviews, we have implemented the FQE method due to its good performance and similarity to the proposed method.
The solutions offered to support LIKE operator in recent articles such as [30] have been towards the use of fuzzy methods and do not have 100% accuracy to provide results. Therefore, we do not select them as methods to compare with our method. The ZSDB results are 100% consistent with the results of a plain LIKE operator, and this feature is not seen in any of the recent fuzzy methods.

In order to evaluate and compare the performance of ZSDB and FQE methods, tests were performed on three computers with a 64-bit operating system, 8 GB of RAM, and an Intel Core i5-3470 processor. We implemented Operators on *Server1* and *Server2* as functions in PostgreSQL. In addition to PostgreSQL, the Java programming language was used to implement data owner protocols. From the performance point of view, evaluation can be evaluated from two aspects of runtime and memory cost. Furthermore, the dataset used in this evaluation is *l_comment* column of line item table contained in the TPC-H data set.

6.1 Memory and Storage Overhead

Fig. 4 shows the memory and storage overhead in ZSDB and FQE methods. In the FQE, each word with length *n* is converted into a term with length $n\ (n + 1) / 2$ and the storage increases with a Quadratic-rate. Additionally, in this method, another column is added to the table. Therefore, in large tables with a long word's length, a high storage overhead is applied to the system. However, in ZSDB, each statement with length *n* turns into a word of length $2(n-2)$. As a result, we are faced with linear growth (the minimum possible value) in ZSDB.

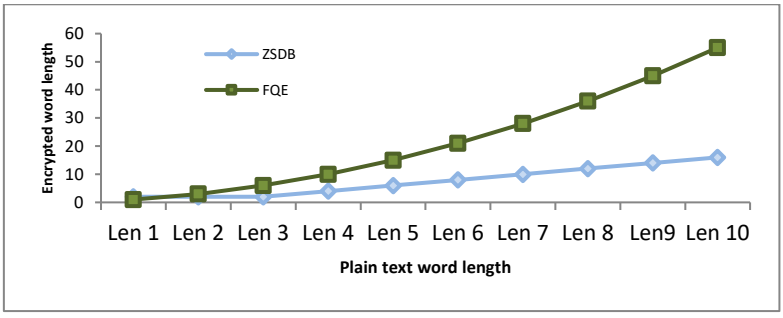


Fig. 4. Memory and Storage Overhead

6.2 Execution Time

We compare ZSDB and FQE performance in terms of the duration of encryption, decryption, and search. Since, in both methods the sentences are encrypted word-by-word, we put the scale in the length of the words and evaluate the length of the words from the variable of a character to ten characters. Moreover, in order to achieve a precise value, we have run each case 1000 times. The result of each run is equal to another 1000 times of the execution.

6.2.1 Encryption

Encryption is a process in which a plain text is converted to an encrypted text. Here, this process is performed by two parts, *Server1* and *Client*. **Fig. 5** shows the encryption execution time in ZSDB and FQE on different word lengths.

In **Fig. 5**, the rectangular bars represent the total execution time, and each line chart refers to a runtime on the data *Client* side or *Server1's* side. As shown, FQE increases with a Quadratic-rate and ZSDB method as a linear rate. The growth of FQE is due to the division of the word into all available subcategories as well as encryption in two different types of encryption.

As a result, as shown in **Fig. 5**, FQE method spends a lot of time on data encryption. For example, FQE method takes 5 ms for ten-letter words, while ZSDB encrypts the word for only 2 ms.

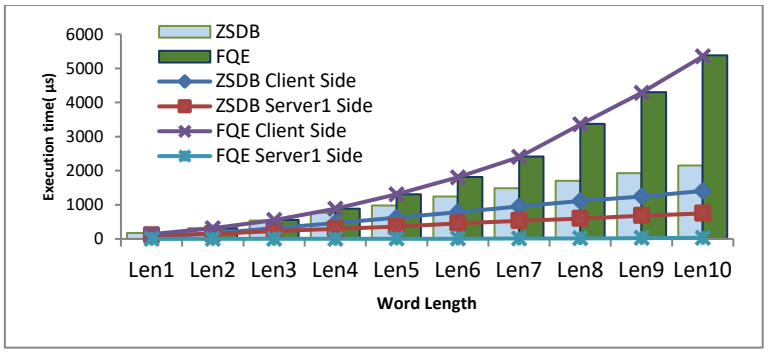


Fig. 5. Encryption Execution Time

6.2.2 Decryption

The time required to convert the encrypted text into plain text in ZSDB and FQE is demonstrated in **Fig. 6**. Based on results, decryption in FQE is fixed. However, in ZSDB, the runtime value is increased linearly. The reason is that FQE has added an additional column to restore the results. Therefore, at decrypting time, FQE only spends the constant time to decrypt the corresponding column. In ZSDB, this amount of time varies. Since words with a length of one, two and three letters are mapped to only one statement, a constant time is also used to decode them. In the next step, as the word length increases, the decoding time also increases linearly.

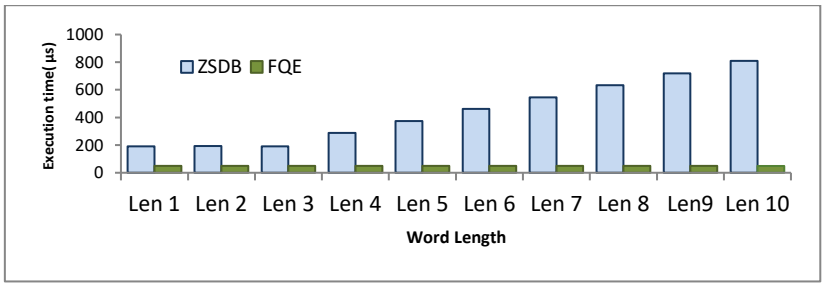
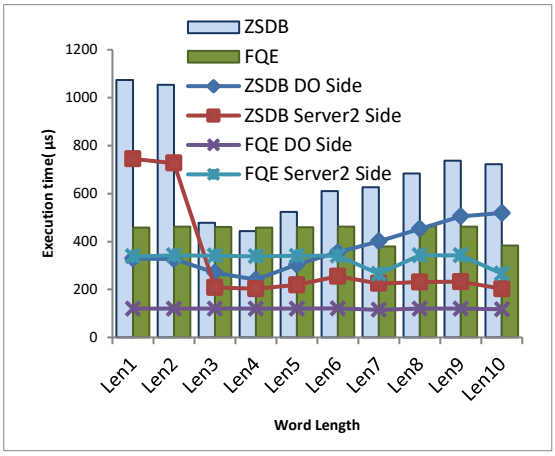


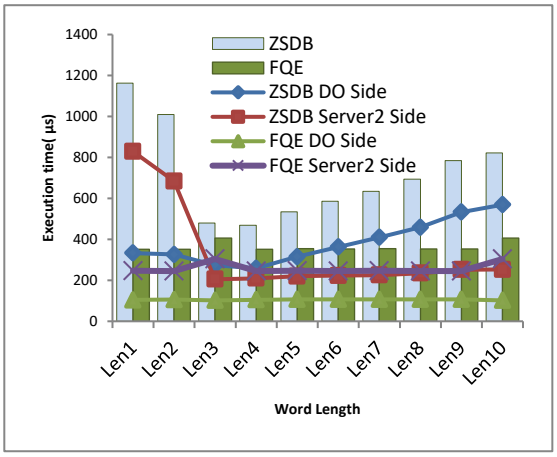
Fig. 6. Decryption Execution Time

6.2.3 Searching

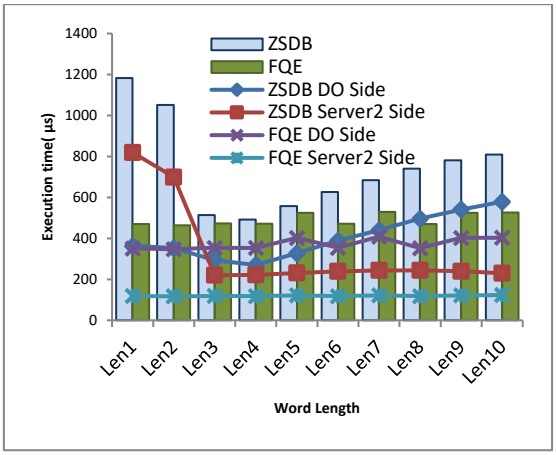
To evaluate the searching execution time, we consider three types of searches, including substring, prefix, and postfix searches on words with different lengths. The results are shown in **Fig. 7**.



(a) The runtime of the suffix search



(b) The runtime of the prefix search



(c) The runtime of the substring search

Fig. 7. Search Execution Time

As shown in **Fig. 7**, the runtimes of searches in FQE, are almost the same. In FQE, the searches are done by encrypting the words and direct compare on the column. However, ZSDB encryption method has multiple steps. Therefore, in ZSDB, the length of words has significant effects on the searches. In a search for substrings, if the word has one or two characters, all its children must be requested from *Server1*. Also, in prefix and postfix searches, all the left and right children should be requested, respectively. Hence, in **Fig. 7** the maximum execution times are related to words with lengths of one or two. The maximum costs of other words with different lengths are closed to each other. As shown in the line charts, the cost of searches on *Server2* is the same.

6.3 Overall Performance

In this section, considering all the execution times obtained from each method, we compare the performance of ZSDB and FQE with plain (non-cryptographic) mode. To calculate performance, we first consider the performance of the plain mode as 1 and the performance of each method as relative to the amount of execution overhead they have. We first execute a read-only workload (Δ=0.0) to measure decryption performance of each method. Next, we evaluate both medium read-write (Δ=0.5) and write-only (Δ=1) workloads to measure hybrid and encrytion performance of each method. **Fig. 8** shows the performance of the methods at different stages.

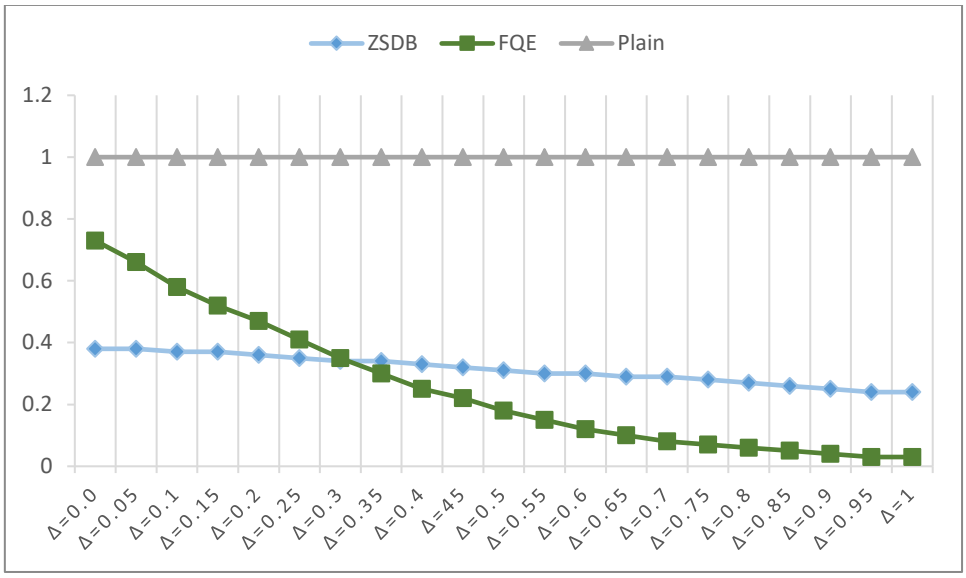


Fig. 8. Performance of the Methods at Different Stages

As **Fig. 8** displays, by increasing write workload, performance of both ZSDB and FQE methods decreases due to the high overhead of encryption, but the ZSDB method has a less decreased performance than FQE. Finally, considering all the execution modes, the performance comparison of the methods is given in **Fig. 9**. Our results show that Overall ZSDB has been able to improve performance, but of course, more improvements are needed to achieve desired performance.

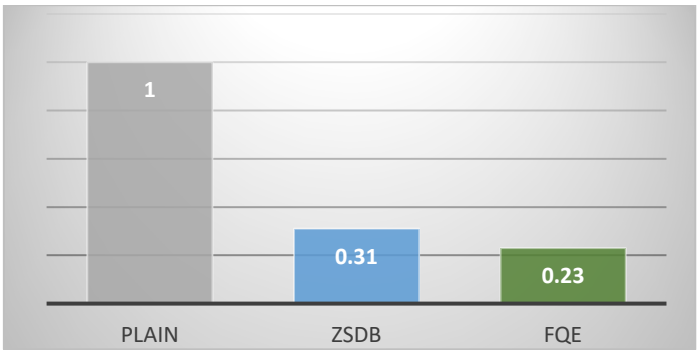


Fig. 9. Overall Performance of the Methods

7. Conclusion

In this paper, a secure query processing method was proposed for a variety of data types. We attempt to consider the security and performance aspects simultaneously.
Despite the other methods, we used secret sharing to support various operations on different data types and avoid the overhaul of homomorphic methods. In the proposed method, we were able to respond to complex query requests by upgrading the SDB method. User's requests can consist of all types of operators, including computational, comparisons and LIKE operators. In this work, we search string data using an additional server (Dictionary server). In this way, we were able to answer a variety of string matching (%Str%, Str%, %Str). In the proposed method, most processes and storage are performed on the servers, and the data owner is only responsible for data encryption, query translation and decrypting the results.

References

[1] J. Tang, Y. Cui, Q. Li, K. Ren, J. Liu and R. Buyya, "Ensuring security and privacy preservation for cloud data services," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, pp. 13, 2016. Article (CrossRef Link).
[2] F. A. Aljumah, *Protocols for Secure Computation on Privately Encrypted Data in the Cloud*, Doctoral dissertation, Concordia University, Canada, 2017. Article (CrossRef Link).
[3] C. Sahin and A. El Abbadi, "Data security and privacy for outsourced data in the cloud," in *Proc. of IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 1731-1734, 2018. Article (CrossRef Link).
[4] L. Bouganim and Y. Guo, *Database encryption. Encyclopedia of Cryptography and, 2nd Edition*, Springer US, pp. 307-312, 2011. Article (CrossRef Link).
[5] P. Singh and K. Kaur, "Database security using encryption," in *Proc. of IEEE International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pp. 353-358, February, 2015. Article (CrossRef Link).
[6] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. of the forty-first annual ACM symposium on Theory of computing*, pp. 169-178, May, 2009. Article (CrossRef Link).
[7] R. L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, 1978. Article (CrossRef Link).
[8] A. C. Yao, "Protocols for secure computations," in *Proc. of 23rd annual symposium on foundations of computer science*, pp. 160-164, November, 1982. Article (CrossRef Link).

[9] 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 *Proc. of the 2014 ACM SIGMOD international conference on Management of data*, pp. 1395-1406, June, 2014. Article (CrossRef Link).
[10] H. Hacigümüş, B. Iyer, C. Li and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proc. of the 2002 ACM SIGMOD international conference on Management of data*, pp. 216-227, June, 2002. Article (CrossRef Link).
[11] E. Mykletun and G. Tsudik, "Aggregation queries in the database-as-a-service model," in *Proc. of the annual conference on data and applications security and privacy*, pp. 89-103, July, 2006. Article (CrossRef Link).
[12] R. A. Popa, C. Redfield, N. Zeldovich and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *Proc. of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 85-100, October, 2011. Article (CrossRef Link).
[13] S. Tu, M. Kaashoek, S. Madden and N. Zeldovich, "Processing analytical queries over encrypted data," in *Proc. of the VLDB Endowment*, pp. 289-300, March, 2013. Article (CrossRef Link).
[14] Z. He, W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, S. M. Yiu and E. Lo, "SDB: a secure query processing system with data interoperability," in *Proc. of the VLDB Endowment*, vol. 8, no. 12, pp. 1876-1879, 2015. Article (CrossRef Link).
[15] K. G. Ho, L. Vu, N. H. Nguyen and H. M. Nguyen, "Speed up querying encrypted data on outsourced database," in *Proc. of the 2017 International Conference on Machine Learning and Soft Computing*, pp. 47-52, 2017. Article (CrossRef Link).
[16] P. G. Alves and D. F. Aranha, "A framework for searching encrypted databases," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1, 2018. Article (CrossRef Link) .
[17] S. S. Moghadam, J. Darmont and G. Gavin, "Enforcing privacy in cloud databases," in *Proc. of the International Conference on Big Data Analytics and Knowledge Discovery*, pp. 53-73, August, 2017. Article (CrossRef Link) .
[18] T. K. Saha, M. Rathee and T. Koshiba, "Efficient private database queries using ring-LWE somewhat homomorphic encryption," *Journal of Information Security and Applications*, vol. 49, no. 1, 2019. Article (CrossRef Link).
[19] R. Pontes, M. Pinto, M. Barbosa, R. Vilaça, M. Matos and R. Oliveira, "Performance trade-offs on a secure multi-party relational database," in *Proc. of the Symposium on Applied Computing*, pp. 456-461, April 2017. Article (CrossRef Link).
[20] Y. Zhou and L. M. Wang, "Sds2: Secure data-sharing scheme for crowd owners in public cloud service," in *Proc. of the IEEE Second International Conference on Data Science in Cyberspace (DSC)*, pp. 22-29, June, 2017. Article (CrossRef Link).
[21] M. He, J. Zhang, G. Zeng and S. M. Yiu, "A Privacy-Preserving Multi-Pattern Matching Scheme for Searching Strings in Cloud Database," in *Proc. of the 15th Annual Conference on Privacy, Security and Trust (PST)*, pp. 293-299, August 2017. Article (CrossRef Link).
[22] B. Wang, S. Yu, W. Lou and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. of the IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 2112-2120, April 2014. Article (CrossRef Link).
[23] D. Wang, X. Jia, C. Wang, K. Yang, S. Fu and M. Xu, "Generalized pattern matching string search on encrypted data in cloud systems," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*, pp. 2101-2109, April, 2015. Article (CrossRef Link).
[24] Z. Liu, J. Li, C. Jia, J. Yang and K. Yuan, "SQL-based fuzzy query mechanism over encrypted database," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 10, no. 4, pp. 71-87, 2014. Article (CrossRef Link).
[25] Z. Wu, G. Xu, C. Lu, E. Chen, F. Jiang and G. Li, "An effective approach for the protection of privacy text data in the CloudDB," *World Wide Web*, vol. 21, no. 4, pp. 915-938, 2018. Article (CrossRef Link).
[26] V. Attasena, N. Harbi and J. Darmont, "A novel multi-secret sharing approach for secure data warehousing and on-line analysis processing in the cloud," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 11, no. 2, pp. 22-43, 2015. Article (CrossRef Link).

[27] W. Tang, B. Qin, Y. Li and Q. Wu, "Functional Privacy-preserving Outsourcing Scheme with Computation Verifiability in Fog Computing," *KSII Transactions on Internet & Information Systems*, vol. 14, no. 1, pp. 281-298, 2020. Article (CrossRef Link).
[28] S. Papadopoulos, A. Kiayias and D. Papadias, "Secure and efficient in-network processing of exact SUM queries," in *Proc. of the 27th International Conference on Data Engineering*, pp. 517-528, April, 2011. Article (CrossRef Link).
[29] S. Fatima and S. Ahmad, "An Exhaustive Review on Security Issues in Cloud Computing," *KSII Transactions on Internet & Information Systems*, vol. 13, no. 6, pp. 3219-3237, 2019. Article (CrossRef Link).
[30] J. Hua, Y. Liu, H. Chen, X. Tian and C. Jin, "An enhanced wildcard-based fuzzy searching scheme in encrypted databases," *World Wide Web*, vol. 23, pp. 2185-2214, 2020. Article (CrossRef Link).

Shahrzad Azizi received her MSC Software Engineering from the University of Zanjan. Her research is focused on testing NoSQL database security, testing database attacks, providing security solutions and investigating innovations in information extraction in Natural Language Processing (NLP).

Davud Mohamadpur received his Ph.D. from the Malek Ashtar University of Technology and is currently an Assistant Professor in the University of Zanjan. His research is focused on improvement of NoSQL, NewSQL databases and Big Data processing frameworks.