Journal of Information Systems and Telecommunication
Vol.14, No.1, January-March 2026, 50-58
http://jist.acecr.org
ISSN 2322-1437 / EISSN:2345-2773

KSDB: Improving Cloud Database Security by Using Searchable Encrypted Data

Davud Mohammadpur[1*], Mahmood Khoeini[1]

[1].Department of Computer Engineering, University of Zanjan, Zanjan, Iran

Received: 19 July 2025/ Revised: 20 February 2026/ Accepted: 13 May 2026

Abstract

Data encryption is a highly effective means of ensuring data security. It transforms readable data into a ciphertext format using cryptographic algorithms and keys. However, the challenge arises when performing query operations on encrypted data due to the alteration of the data structure. This article introduces an improved method that facilitates encryption and query operations on encrypted cloud data without requiring decryption. By leveraging reverse indexing, information mapping, and secret sharing across multiple servers, the proposed method KSDB guarantees data security and prevents data disclosure during both the encryption and query execution processes. The KSDB is an application-level encryption technique that the encrypted data is stored in the cloud storage. While existing methods primarily concentrate on numerical data, this study places emphasis on maintaining the confidentiality of string data, enabling search operations on partial strings without decryption. The results and evaluations demonstrate a significant reduction in memory consumption achieved by the proposed method. In KSDB all implementations have been migrated to a dedicated private server. This secure and reliable entity is responsible for managing critical data, including encryption keys. This strategic decision effectively resolves security issues present in pervious methods and facilitates encryption and decryption processes. Furthermore, it not only addresses concerns regarding information leakage but also enhances data confidentiality.

Keywords: Secure Database; Searchable Encryption; Cloud Storage; Secure SQL Query.

1- Introduction

Cloud computing has changed how software, platforms, and storage are provided, making them more flexible and cost-effective for businesses of all sizes. Cloud-based databases have garnered significant attention over the past few decades and become popular in recent years. Key advantages of cloud-based data storage include streamlined information technology infrastructure and management, remote accessibility from any location globally, and cost-effectiveness [1]. Despite these benefits, there exist potential risks such as unauthorized access, data breaches, exposure of sensitive information, and privacy infringements [2, 3]. Consequently, it is imperative to delve deeper into the security and privacy challenges associated with cloud computing. When data is stored on remote servers, users relinquish physical control, transferring it to potentially untrustworthy cloud providers. Therefore, security and privacy concerns are paramount and pervasive in the realm of cloud computing [3, 4].

When data is stored in the cloud, the data owner, typically the data user, no longer has direct ownership of the physical infrastructure. This shift in ownership necessitates the implementation of specialized security methods tailored to cloud platforms, as traditional architectures and security measures may be less effective. Ensuring data security to protect it from intentional and accidental threats is essential, which involves restricting access to unauthorized users while enabling seamless and immediate access for authorized users to the data required for their tasks. Encryption is one way to protect sensitive data; however, it alters the data and causes the encrypted data to lose its structure, making it impossible to use and run queries directly [3, 5, 6]. This article aims to present a method for encrypting data in cloud storage while maintaining data confidentiality, allowing for secure execution of SQL queries on the encrypted data without compromising information security.

✉ **Davud Mohammadpur**
dmp@znu.ac.ir

1-1- Data Encryption

When data is outsourced to the cloud, it becomes vulnerable. Data encryption is an effective technique for protecting data, as it converts the original form of data into a string of directly unreadable codes, known as ciphertext. Encryption can be implemented at three different levels, with the granularity of encryption chosen according to users' needs and types of operations [7, 8].

Storage-level encryption: At this level, data is encrypted in the storage subsystem. This level of encryption is suitable for encrypting files, entire directories, or storage media. Since data is decrypted for database operations at this level of encryption, the encryption strategy cannot be related to data security in databases.

Database-level encryption: This level is used to secure data inserted or retrieved from the database with the help of a specific algorithm. The encryption strategy can be related to data partitions or required accesses, and encryption can be on the part of the database elements. At this level, encryption can be applied selectively in different details such as rows, columns, and tables. Database-level encryption requires changes in the execution algorithms of database operations, so its use in cloud platforms is not considered.

Application-level encryption: At this level, encryption and decryption processes are performed in applications that communicate with the databases independently. Data is sent to the database in encrypted form, stored and retrieved in encrypted form, and finally decrypted in the applications. Different subtleties can be considered in this type of encryption, and the data can be encrypted according to their sensitivity level. The advantage of application-level encryption is the reduction of excessive loads on the database server due to encryption and decryption operations by separating the encryption keys from the encrypted data stored in the database. However, applications must be designed to support encryption and decryption capabilities.

1-2- Searchable Encryption

Many individuals and organizations opt for cloud storage to house their databases due to its expansive storage capacity and adaptable services. As previously mentioned, data owners typically encrypt their data prior to uploading it to the cloud in order to safeguard its confidentiality. However, as the data is encrypted within the cloud, users are unable to directly access the encrypted data. A viable solution to this issue is the utilization of searchable data encryption. Searchable encryption constitutes a cryptographic approach that permits authorized users to search for and retrieve encrypted data in the cloud, including through keyword queries. A pivotal aspect of this method is that the cloud server conducts searches on the encrypted data without possessing knowledge of the encrypted data content, subsequently delivering it to the users.

Regarding encryption, searchable encryption methods can be categorized into two classifications: searchable symmetric encryption (SSE) and public key encryption with keyword search (PEKS) [5]. SSE exclusively allows private key holders to generate ciphertexts and perform decryptions for searches, while PEKS enables multiple users who possess the public key to generate ciphertexts, yet only permits the private key holder to conduct decryptions [9, 10].

1-3- Strings Searchable Encryption

Searchable encryption methods for string data types enable querying of encrypted string data in databases. These methods need to support LIKE operator to search on encrypted data. One of the methods used to support this operator on encrypted data is the utilization of full-text search [8, 9].

Full-text search is an advanced technique for searching in databases using a word index. Unlike primitive text search using the LIKE operator, which matches words in the text at a slower speed, full-text search indexes the location of all words in a text beforehand. This allows users to achieve their search by utilizing the index, eliminating the need to browse through the entire data for each search. As a result, searching through millions of records using the LIKE operator can be time-consuming, whereas a full-text search provides quick results.

The process of full-text search is typically divided into two tasks: indexing and searching. During the indexing phase, the text of the documents is scanned, and a list of search terms is generated. For each term or word found in a document, an entry is created in the index, noting its relative position in the document. In the searching phase, when a specific query is performed based on the created index, all documents containing a word from the query are identified, and all possible results that match the search criteria are returned.

2- Related Works

In this section, we present an overview of the SDB method [11] and subsequently introduce the ZSDB method [8] as the foundational approach in our proposed method.

In secret sharing methods, data is divided into n parts, with each part further divided among multiple servers. The original secret (desired data) can only be reconstructed when all parts are combined; individual parts alone hold no validity. The SDB method employs secret sharing for data encryption at the application level. In SDB, sensitive data is split into two parts to enable secure querying. One part is

stored on the reliable data owner's side, while the other part is stored on the less trustworthy server side. Non-sensitive data is stored in plain text on the server side. Additionally, SDB offers various operators that can be applied to encrypted data, plaintext data, or a combination of both. However, it should be noted that the SDB method exclusively supports integer data types and does not accommodate string types [11].

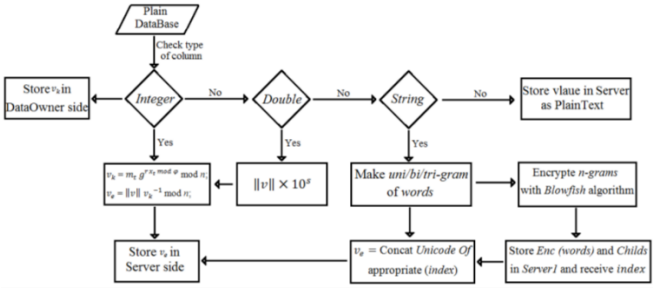


Fig. 1. Execution process of ZSDB [8]

The ZSDB method, also an application-level encryption approach, utilizes the SDB method for encrypting integer data. For decimal numbers, ZSDB first maps them to integers by multiplying them with 10 and then applies the same encryption method. Notably, ZSDB extends its support to string data types as well. ZSDB incorporates a searchable encryption technique that enables users to upload their encrypted data to an insecure server while retaining the ability to search it without exposing the raw data to the server. The execution process of the ZSDB method is illustrated in Fig. 1 [8].

As shown in Fig.1, in the architecture of the ZSDB, a separate server is used to support operators related to string data and store a subset of words. In ZSDB, during the encryption stage, sentences are first divided into words, and for each word, trigram subsets are generated based on its length. Then each of the generated subsets is encrypted using the Blowfish algorithm and stored in the Dictionary server of the respective server. Subsequently, an index is considered for each subset of words in the Dictionary server and sent to the data owner. In the next, the data owner receives the necessary subset indices based on the length of each word and concatenates them to create a sequence of characters considered as a unicode equivalent phrase, which is then stored as an encrypted phrase in another server. In other words, for words based on the indices related to their trigram forms, a new phrase is generated and stored. Moving forward to the search stage, trigram subsets of the desired search word are also generated, and each corresponding index is requested from the Dictionary server. Based on the index values, the unicode phrase of the search word is determined and matched with the unicode phrases stored in another server to execute relevant record searches based on this match [8].

However, it is important to mention that despite providing an appropriate search mechanism, the ZSDB method faces challenges related to information disclosure and the size of the generated index. Specifically, it suffers from information disclosure based on the frequency of search terms in queries sent to the server [8].

3- Proposed Method

In this paper, we propose an improved method called KSDB (Keyword-based Secure Database), which enables efficient encryption for both numeric and string data types. The KSDB method is developed based on modifications and enhancements to the existing ZSDB approach. One of the key differences is that instead of relying on trigrams, KSDB operates directly on the words extracted from text. This modification reduces processing and memory overheads and making it well-suited for practical applications involving heterogeneous data types.

3-1- Architecture and Structure

The architecture of our proposed method is illustrated in Fig. 2. KSDB is an application-level encryption technique based on secret sharing. In this method, the encrypted data is stored in the cloud, while a trusted private server handles key management and performs encryption and decryption operations. This architecture offers several advantages:
User transparency: The details of the encryption process are hidden from the user, and the private server takes responsibility for encryption, decryption, and key maintenance.
Enhanced security: By utilizing multiple servers, symmetric encryption can be employed without concerns about information leakage or key distribution.
Simplified implementation: The use of symmetric encryption eliminates the need for additional complexities such as customer authentication and certificate authority.

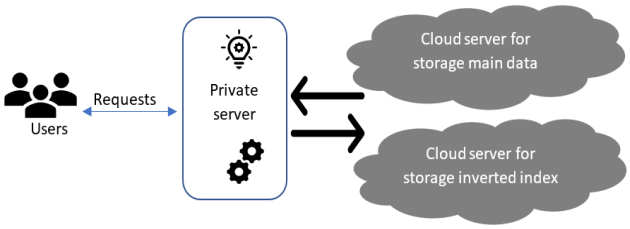


Fig. 2. Architecture of the KSDB method

KSDB utilizes the searchable symmetric encryption (SSE) method for data encryption. In this approach, the search term is encrypted using a key and sent to the cloud database as a search token. The search token is then compared with

the encrypted data using the XOR operation, and the result is returned without revealing the content of the encrypted text. However, SSE has a limitation when it comes to searching for substrings using the LIKE operator. To address this limitation, KSDB incorporates a data indexing step before storing the data in the cloud database. This indexing allows for substring matching and serves as an additional layer of encryption. Instead of using plain text, KSDB employs mapped codes of the indexed data as strings, enabling full-text search capabilities. KSDB can be directly applied in real-world cloud-based systems such as healthcare record management, financial data repositories, and enterprise document storage, where sensitive data must remain encrypted while still supporting efficient keyword and substring search functionalities.

3-2- Details of the Proposed Method

The objective of KSDB is to ensure secure storage and query execution on cloud databases, with a specific focus on string data types. To achieve this, the proposed method encompasses several steps, including the creation of a data dictionary, encryption, storage of an encrypted dictionary, query execution, and decryption. Each of these steps is elaborated upon below.

3-3- Database Servers

In the implementation of KSDB, particular attention is given to the security of string data, while the encryption and decryption operations for integer and decimal data types follow a similar approach as presented in the ZSDB method [8]. As depicted in Fig. 3, the architecture of this method consists of three database servers: a secure local server (referred to as the private server) responsible for storing keys and word lists in plain text, a cloud server (referred to as the first server) that stores the encrypted reverse index list, and another cloud server (referred to as the second server) that stores the central database with encrypted data. It is assumed that these servers are independent of each other and operate separately, allowing for the guarantee of data security and privacy. Each server can have its own separate DBMS.

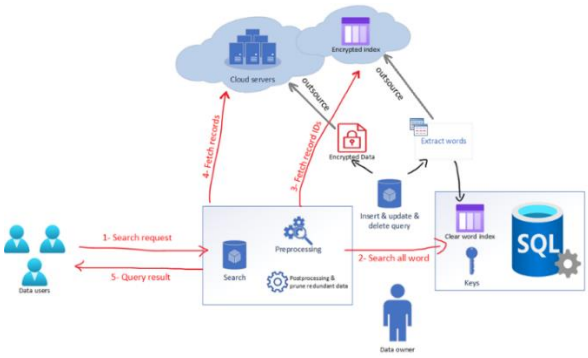


Fig. 3. Handling requests in KSDB method

3-4- Encryption

The encryption process occurs concurrently with any modifications made to the database, such as insertion, editing, or deletion. Initially, when creating tables on the second server, their structure is saved and maintained on the private server. Additionally, separate keys are generated and stored alongside the table structure for each column containing sensitive string data. For each data record, the encryption steps are outlined in Fig. 4.

The value is converted into words for each string column and stored and indexed in the plain text word list table on the private server. Subsequently, the primary data is encrypted using the keys associated with its column and stored along with other data on the second server, returning the desired record number. Finally, a data record is created on the private server for each word of the string data stored in the plain text list. This record includes the coded number of the word stored in the plain text list, the table name, the column name, and the number of the primary data record stored on the second server.

```
Input: plaintext text
Output: record identifier r_id stored in cloud

1:  enc_str ← Encrypt(text)
2:  r_id ← Server2.Insert(enc_str, T_enc)
3:  words ← Tokenize(text)
4:  for each w in words do
5:      idx_id ← PrivateServer.GetOrCreateID(w)
6:      enc_idx ← Encrypt(idx_id)
7:      Server1.UpdateIndex(enc_idx, r_id)
8:  end for
9:  return r_id
```

Fig 4. Pseudocode of string data encryption in KSDB

3-5- Decryption

Referring to Fig. 3, when a query execution request is received from the user, the LIKE operator is first checked to determine if it is being used. If it is not used or if a string matching and comparison operator is employed, the string data is directly encrypted using the keys of the desired column, and the data is retrieved from the second server. Suppose the query includes the LIKE operator; based on whether it is used for substring matching, extension matching, or prefix matching, the search operation retrieves words related to the search from the plain text list. Subsequently, after encrypting these words, the indexes (row numbers) of these words are retrieved from the list of encrypted words stored on the first server.
During query execution, the list of original data records is determined based on which tables and data columns are requested. The intersection of records for each word is obtained from this list, and based on this resulting list, the desired records are requested from the second server. After decrypting the data on the private server, any records that do not contain the text requested by the user (i.e., records that contain the requested words but do not preserve their order) are removed from the result list. Finally, the result is sent to the user. The pseudocode for this operation is provided in Fig. 5.

3-6- Security

In the context of database encryption, it is essential to address the following security requirements [11]:
Isolation of the query: It is imperative from a security perspective that the untrusted server does not gain access to any plaintext data information from the obtained results.
Controlled search: Only authorized users and the data owner should have the capability to query the data.
Concealed queries: User queries must be conducted in a manner that prevents the revelation of any data-related information on the server side.
In the KSDB method, assuming the private server's safety and reliability, the database structure, keys, and word index are stored within it, and both encryption and decryption processes are carried out on this server. Furthermore, as only encrypted information is present on other servers, they are unable to disclose any information.
Additionally, a reverse index structure is housed in the first server, serving as the sole index of the words stored in the private server, with these indexes being stored in an encrypted format. The encrypted data is exclusively stored on the second server, rendering information disclosure impossible even in the event of server collusion.
Another potential attack vector to consider is frequency analysis and data inference. In this type of attack, the frequency of words in various texts is analyzed. In the KSDB method, all string data undergoes encryption in a single step using distinct keys (record keys), effectively preventing attackers from gaining any insight into word frequency. Consequently, this method effectively addresses the second requirement. With regard to the third requirement, which pertains to concealing queries, the KSDB method ensures that search requests are transmitted to the cloud servers in an encrypted form for execution by the data owner. Consequently, the servers remain unaware of the content being searched.

```
Input: query string q
Output: matching plaintext records

words ← Tokenize(q)
token_list ← ∅
for each w in words do
    idx_id ← PrivateServer.GetID(w)
    if idx_id ≠ -1 then
        t ← Encrypt(idx_id)
        token_list ← token_list ∪ { t }
    end if
end for
enc_record_set ← ∅
for each t in token_list do
    Rt ← Server1.SearchIndex(t)
    enc_record_set ← enc_record_set ∪ {Rt}
end for
C_list ← Server2.FetchEncrypted(enc_record_set)
result ← ∅
for each c in C_list do
    m ← Decrypt(c)
    if SubstringMatch(m, q) then
        result ← result ∪ {m}
    end if
end for

return result
```

Fig. 5- Pseudocode of string data decryption KSDB

4- Results and Evaluation

The KSDB method employs the ZSDB method to encrypt numeric data types. By extending the SDB method, which was initially limited to encrypting integers, the ZSDB method incorporates encryption for decimal numbers and string data. Previous evaluations have confirmed the versatility of the SDB method in supporting various operators and producing reliable results [12, 13, 14]. Thus, this section focuses solely on evaluating the KSDB method for string data type. Fuzzy methods that support the LIKE operator, although not chosen for comparison due to their lack of accuracy in presenting results [15], are based on

fuzzy techniques for string data. In contrast, the KSDB method delivers results that perfectly match those obtained using a simple LIKE operator, a feature absent in recent fuzzy methods.

To compare the proposed method for string data, we implemented ZSDB under identical conditions as the KSDB method. Subsequently, we analyzed the results in terms of storage overhead and execution time cost, as presented below.

To evaluate the efficiency of both ZSDB and KSDB methods, we conducted executions on three computers equipped with an Intel Core i7 2.7GHz processor, 32GB DDR4 memory, and NVMe type SSD storage memory. PostgreSQL databases were used across all three servers, while Python programming languages facilitated the execution process. In our evaluations, we utilized id (integer data type) and product_title_fa (string data type) columns from the Digikala Products dataset.

4-1- Memory Usage

Memory usage differs between the ZSDB and KSDB methods. In the ZSDB method, each phrase of length n is converted into (n-2) words, resulting in a storage size of 2(n-2) when mapped to encrypted data. On the other hand, the KSDB method encrypts all data using the Advanced Encryption Standard (AES) algorithm with a 32-character key. If each phrase has a length of n, the length of the encrypted text is 2n. Both methods exhibit linear growth. The KSDB method utilizes a word index to store word information, including word placement within encrypted records. Conversely, the ZSDB method employs two lists: one for word indexing and another for generating 3-grams containing n-2 words that are unnecessary in KSDB. Fig. 4 illustrates that KSDB only requires 8% of ZSDB's memory space for storing encrypted data on the server, thereby KSDB reducing memory overhead by 92%.

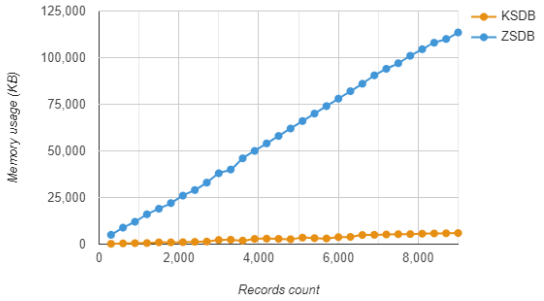


Fig. 4. Comparison of the memory usage

4-2- Encrypting Execution Time

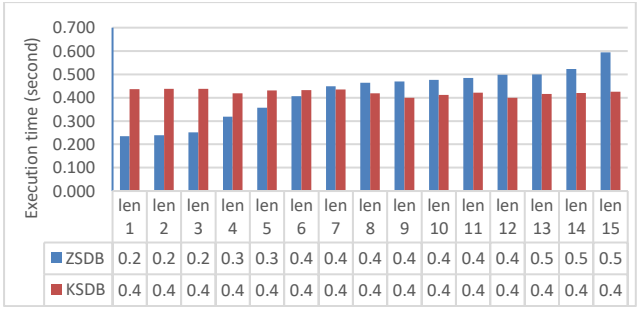
Execution time refers to the duration required to perform a specific action. In this study, we compare the execution time of the KSDB and ZSDB methods for data insertion (encryption) in the database, as well as data searching and retrieval (decryption) from the database.

The encryption execution time is influenced by the time cycle involved in inputting plain text into the program, encrypting it, and inserting it into the database. Since both methods divide string values into words and then encrypt them, the length of the words can impact the execution time. To evaluate the execution time, we conducted 15 steps, each consisting of ten records with a string of 10 words. The word lengths in these steps ranged from one to 15 letters. The results of the execution times for both methods are presented in Fig. 6.

In the ZSDB method, the encryption time is affected by the length of the words. As depicted in Fig. 6, the encryption time remains constant for words with a length of one to three letters and then increases linearly. On the other hand, in the KSDB method, the required time is almost constant because encryption is performed in a single step.

Furthermore, to compare the total encryption execution time, we evaluated both methods using the DigiKala Products dataset. This evaluation involved measuring the duration of encryption operations for various record counts (up to 9000 records). The comparison between the two methods is illustrated in Fig. 7.

As shown in Fig. 7, the encryption execution time of KSDB is approximately 34% higher than that of the ZSDB method. This difference in execution time arises because ZSDB aims to transfer most calculation processes to the server side, minimizing the computational overhead on the data owner's side. In contrast, KSDB performs all executive functions on the private server side to address concerns about server security and privacy.



	len 1	len 2	len 3	len 4	len 5	len 6	len 7	len 8	len 9	len 10	len 11	len 12	len 13	len 14	len 15
ZSDB	0.2	0.2	0.2	0.3	0.3	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.5
KSDB	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4

Fig. 6. Encryption execution time comparison for different word lengths

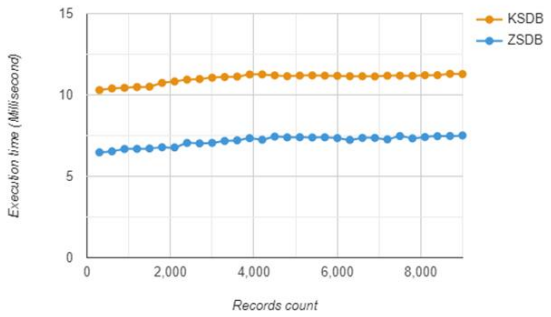


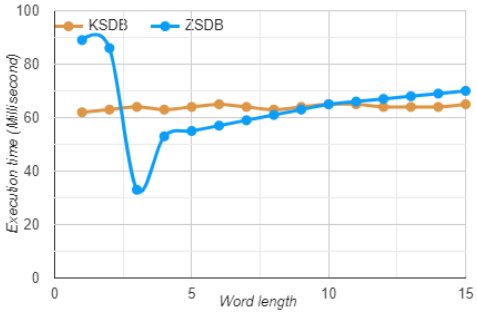
Fig. 7. Encryption execution time comparison for each record

4-3- Decrypting Execution Time

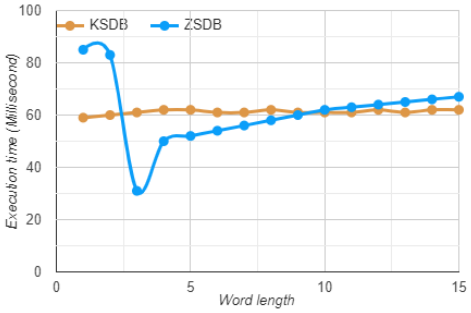
Moving on to decrypting execution time, this refers to the time required to convert encrypted text back into plain text. In the ZSDB method, similar to encryption, the word length also affects decrypting time. Consequently, decrypting time remains almost constant for words with a length of one to three letters and then increases linearly. However, in the KSDB method, since encryption and decryption occur simultaneously (as mentioned in the data encryption section), the required time remains almost constant. It is worth noting that evaluating this section involves decrypting and searching for expressions within encrypted information, which necessitates possession of the key and decryption parameters related to the desired column.

To evaluate execution time during search operations, we considered three modes: substring search, prefix search, and suffix search, each with different word lengths. These modes are visualized in Fig. 8. The evaluation process for execution time begins when the user submits their request. The processing operation is then performed on the query, resulting in a suitable query for retrieving data from the encrypted data server. Subsequently, the query is sent to the encrypted data server, where records are fetched and an appropriate result is displayed to the user.

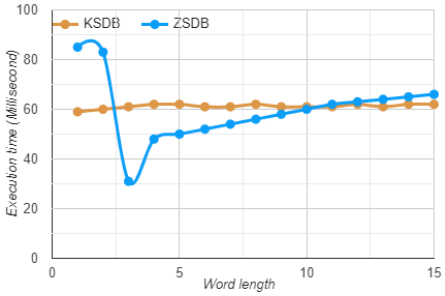
As depicted in Fig. 8, the search execution time in KSDB remains relatively constant within a specific interval across all three modes. This method conducts the entire process on a private server at the program level and involves post-processing operations after fetching records, resulting in a same average execution cost compared to the ZSDB method.



a. Substring search execution time



b. Prefix search execution time



c. Suffix search execution time

Fig. 8. Search execution time

Conversely, in ZSDB, the search process cost increases linearly with the number of characters due to its dependency on word length. Notably, for single-character and two-character words in the ZSDB method, querying all the children of the words from the encrypted data server during the search contributes to longer execution times. This is influenced by the number of children and sub-branches of one- and two-character terms, leading to extended search tree checking for these terms.

5- Security Analysis and Threat Model

In the original architecture, the private server is assumed to be fully trusted. To strengthen the security analysis, we consider a stronger adversarial model including:

- Honest-but-curious cloud server
- Malicious cloud server
- Insider threat in the private server
- Key compromise attacker

The cloud server stores encrypted data and indexed structures, while the private server manages keys and performs encryption/decryption and search token generation.

5-1- Cloud-Side Security

KSDB relies on Searchable Symmetric Encryption (SSE), where data is encrypted before outsourcing, and search tokens are generated using a secret key. The cloud performs matching over encrypted data without accessing plaintext. Under standard SSE security assumptions, the scheme guarantees data confidentiality while allowing controlled leakage limited to search and access patterns. To reduce leakage impact, periodic re-keying and re-indexing can be applied. The indexing and mapped-code mechanism used for substring search provides an additional obfuscation layer, as no plaintext substrings are stored in the cloud.

5-2- Insider Threats and Key Compromise

The most critical security risk in KSDB arises if the private server is compromised. Since this server is responsible for key management and cryptographic operations, unauthorized access to its cryptographic keys could threaten data confidentiality. In particular, disclosure of master or encryption keys may enable an attacker to generate valid search tokens or decrypt stored data.

To reduce this risk, KSDB can employ threshold secret sharing for master keys so that compromising a single component does not reveal the full key. In addition, separating data encryption keys from search-token generation keys limits the impact of partial key exposure. Storing keys in hardware-backed secure modules further prevents direct key extraction, while periodic key rotation and audit logging of cryptographic operations help detect and contain potential misuse. As long as fewer than the required number of secret shares are compromised, no meaningful information about the master key can be reconstructed.

5-3- Security Argument

The confidentiality of KSDB depends on three main components: the semantic security of the underlying symmetric encryption scheme, the controlled leakage properties of the searchable symmetric encryption (SSE) construction, and the threshold security provided by the secret sharing mechanism. Together, these components ensure that data remains protected even when stored and queried in an outsourced environment.

Therefore, unless an adversary is able to reconstruct the required number of secret shares or completely extract the protected master keys, recovering the original plaintext data remains computationally infeasible. Under standard cryptographic assumptions, any attacker without sufficient key material gains no practical advantage in distinguishing or decrypting the stored information.

6- Conclusions

In developing the encryption method, our objective was to support various data types while upholding confidentiality. To achieve this, the proposed method leverages the numerical data encryption strength of the ZSDB method as a foundation, with modifications to the string data encryption method and the incorporation of the SSE algorithm alongside an additional indexing layer. This approach effectively addresses information leakage concerns during substring searches.

Our evaluation of the proposed method primarily focuses on enhancing efficiency and security. The results from these analyses indicate that in the private server implementation of KSDB, there is an approximate 34% increase in overhead during data insertion compared to ZSDB. Both methods require a similar amount of time for searching and extracting data. However, the removal of the tree structure in KSDB has resulted in a significant reduction of approximately 92% in additional memory overhead. Moreover, to ensure data security and prevent potential information breaches, all KSDB implementations have been migrated to a dedicated private server. This secure and reliable entity is responsible for managing critical data, including encryption keys. This strategic decision effectively resolves security issues present in the ZSDB method and facilitates encryption and decryption processes.

References

[1] H. Tabrizchi, and M. Kuchaki Rafsanjani, “A survey on security challenges in cloud computing: issues, threats, and solutions”, The journal of supercomputing, Vol. 76, No. 12, 2020, pp. 9493-9532.

[2] N. Mohammadi, A. Rezakhani, and H. Haj Seyyed Javadi, “FLHB-AC: federated learning history-based access control using deep neural networks in healthcare system”, Journal of Information Systems and Telecommunication (JIST), Vol.12, No. 46, 2024, pp. 90-104.

[3] V. Govindarajan, "A Novel System for Managing Encrypted Data Using Searchable Encryption Techniques", International Journal of Advanced Computer Science & Applications, Vol. 16, No. 3, 2025, pp. 22-34.
[4] L. Rikhtechi, V. Rafe, and A. Rezakhani, "Secured access control in security information and event management systems", Journal of Information Systems and Telecommunication (JIST), Vol. 9, No. 33, 2021, pp. 67-78.
[5] U. Butt, R. Amin, M. Mehmood, H. Aldabbas, M. Alharbi, and N. Albaqami, "Cloud security threats and solutions: A survey", Wireless Personal Communications, Vol. 128, No. 1, 2023, pp. 387-413.
[6] O. Ebadati, F Eshghi, and A Zamani, "Security enhancement of wireless sensor networks: A hybrid efficient encryption algorithm approach", Journal of Information Systems and Telecommunication (JIST), Vol. 6, No. 23, 2018, pp. 180-192.
[7] C. Wang, K. Ren, W. Lou, and J. Li, "Toward publicly auditable secure cloud data storage services", IEEE Network, Vol. 24, No. 4, 2010, pp. 19-24.
[8] S. Azizi, and D. Mohammadpur, "Searchable Encrypted String for Query Support on Different Encrypted Data Types", KSII Transactions on Internet & Information Systems, Vol. 14, No. 10, 2020, pp. 4198-4213.
[9] E. Khalaf, and M. Kadi, "A survey of access control and data encryption for database security", Journal of King Abdulaziz University, Vol. 28, No. 1, 2017, pp. 19-30.
[10] S. K. Kermanshahi, J. K. Liu, R. Steinfeld, S. Nepal, S. Lai, R. Loh, and C. Zuo, "Multi-client cloud-based symmetric searchable encryption", IEEE Transactions on Dependable and Secure Computing, Vol. 18, No. 5, 2019, pp. 2419-2437.
[11] 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", VLDB Endowment, Vol. 8, No. 12, 2015, pp. 1876-1879.
[12] 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, 2014, pp. 1395-1406.
[13] E. Bertino, and R. Sandhu, "Database security-concepts, approaches, and challenges", IEEE Transactions on Dependable and secure computing, Vol. 2, No.1, 2005, pp. 2-19.
[14] Q. Wang, D. Hu, M. Li, Y. Q, "Secure Multi-Character Searchable Encryption Supporting Rich Search Functionalities", IEEE Transactions on Knowledge and Data Engineering, Vol. 38, No. 3, 2026, pp. 1958-1972.
[15] Z. Liu, J. Li, J. Li, C. Jia, J. Yang, and K. Yuan, "SQL-based fuzzy query mechanism over the encrypted database", International Journal of Data Warehousing and Mining (IJDWM), Vol. 10, No. 4, 2014, pp. 71-87.