Received: 15 March 2023 | *Revised:* 6 May 2023 | *Accepted:* 12 May 2023
DOI: 10.37917/ijeee.20.1.7 *Early View* | *June 2024*

Iraqi Journal for Electrical and Electronic Engineering
Original Article

Open Access
IJEEE University of Basrah College of Engineering

Improving Performance of Searchable Symmetric Encryption Through New Information Retrieval Scheme

Aya A. Alyousif*, Ali A. Yassin
Department of Computer science, Education College for Pure Sciences, University of Basrah, Basrah, 61004, Iraq

Correspondance
*Aya A. Alyousif
Department of Computer Science
Education College for Pure Sciences
University of Basrah, Basrah, 61004, Iraq
Email: pgs.aya.alyousif@uobasrah.edu.iq

Abstract
Searchable symmetric encryption (SSE) is a robust cryptographic method that allows users to store and retrieve encrypted data on a remote server, such as a cloud server, while maintaining the privacy of the user's data. The technique employs symmetric encryption, which utilizes a single secret key for both data encryption and decryption. However, extensive research in this field has revealed that SSE encounters performance issues when dealing with large databases. Upon further investigation, it has become apparent that the issue is due to poor locality, necessitating that the cloud server access multiple memory locations for a single query. Additionally, prior endeavors in this domain centered on locality optimization have often led to expanded storage requirements (the stored encrypted index should not be substantially larger than the original index) or diminished data retrieval efficiency (only required data should be retrieved).we present a simple, secure, searchable, and cost-effective scheme, which addresses the aforementioned problems while achieving a significant improvement in information retrieval performance through site optimization by changing the encrypted inverted index storage mechanism. The proposed scheme has the optimal locality O(1) and the best read efficiency O(1)with no significant negative impact on the storage space, which often increases due to the improvement of the locality. Using real-world data, we demonstrate that our scheme is secure, practical, and highly accurate. Furthermore, our proposed work can resist well-known attacks such as keyword guessing attacks and frequency analysis attacks.

Keywords
Information Retrieval, Inverted Index, Locality, Searchable Encryption.

I. Introduction

Cloud storage is a type of online storage that allows users to store, access, and manage their data and files over the Internet, without the need for physical storage devices such as hard drives or USBs. This technology has gained a lot of attention recently due to the many advantages which offers to individuals and businesses alike [1], [2]. One of the primary advantages of cloud storage is its accessibility and availability. With cloud storage, users can access their files from anywhere/anytime in the world, another advantage of cloud storage is its scalability, with cloud storage, users can easily increase or decrease their storage capacity as needed, without the need to purchase additional hardware. Cloud storage providers typically implement strict security measures to safeguard customer data, such as encryption, secure data centers, and access controls. However, the data owner (DW) or company is ultimately responsible for ensuring the protection of sensitive data and compliance with relevant regulations or industry standards.

One important operation that can be applied to sensitive data in the cloud that is searchable symmetric encryption (SSE) [3]. It is a cryptographic method that enables encrypted data to be searched in a secure and efficient manner. It allows data to be encrypted before storage on the cloud server, while still allowing authorized users to search, retrieve and use the data

without compromising its security. In contrast to traditional encryption schemes, where data is encrypted and remains inaccessible until decrypted with a secret key, SSE encrypts data using a symmetric key and creates an additional index for efficient searching. This index is also encrypted, ensuring that the underlying data remains secure. SSE is particularly useful in situations where data privacy is crucial, such as in healthcare, finance, and other industries where sensitive data is stored and accessed. By utilizing SSE, users can store and search their data securely, without the need to expose their sensitive information to unauthorized parties or third-party service providers.

In more detail, SSE involves the DW encrypting the data using a secret key as the initial step, followed by transmitting it to the remote server. Subsequently, the DW creates a secure index from its own database. The encrypted data and secure index are then transmitted to the cloud storage server, which may be either the same untrusted server or a third-party server chosen by the DW. The DW generates a search token that is utilized to retrieve information from a secure index file to execute a secure search.

One of the challenges faced by SSE is efficiency and scalability. The larger the size of the database, the greater the size of the index, and thus the slower retrieval process. After many researches in this field, it was found that this vulnerability is not due to encryption operations, but rather because of the nature of storing the encrypted index, which leads to visiting the cloud server to multiple memory locations in the search process to answer a single query, this is called poor locality [3–7]. This situation results in a significant performance weakness that deteriorates with an increase in database size. Consequently, some researchers have redirected their attention towards improving locality. Although enhancing locality can improve system performance, it often negatively impacts other key characteristics, such as reading efficiency (The server should return only the encrypted data requested from it and no additional data) or storage capacity (A very large increase in the storage space of the encrypted index). As a result, developing a scheme that balances optimal locality with efficient reading and storage poses a significant challenge.

In this study, we present an optimal locality secure scheme to address the challenge of searchable encryption in large datasets and to increase the performance of information retrieval by improving the locality through a change in the encrypted inverted index storage mechanism. Our contributions can be summarized as follows:

- The performance of information retrieval, for both large and small databases, is significantly improved as a result of the enhanced locality.
- Optimal locality O(1). The cloud server will only need to visit one memory location, rather than multiple locations, to respond to the user's query.
- Best read efficiency O(1). The cloud server responds with only the requested data as a result of the user's query.
- The proposed scheme is highly secure as the server searches for and sends the requested data to the data owner without decrypting it.
- This enhances the resistance to various attacks that searchable symmetric encryption is susceptible to.

The remaining sections of this paper are as follows: Section II displays primitive tools, Section III covers previous related works, Section IV presents the scheme, while Section V provides a detailed discussion of the scheme. Section VI contains the security analysis of our scheme, Section VII presents the experimental results, and finally, in the last section, Section VIII, the paper concludes.

II. PRIMITIVE TOOLS

A. SSE algorithms

All algorithms are performed among three fundamental components: data owner(DW), cloud server (CS), and users(U_i), where $i \in N_u$ and N_U represents the total number of users in the system.

SSE consists of the following algorithms[3], [5].

- $k \longleftarrow Gen(1^\lambda)$: The data owner DW runs a key generation algorithm to generate a secret key k by providing a security parameter 1^λ as input.
- $SI \longleftarrow Enc(k, DB)$: Is used by DW for create a secure index file (SI) based on a secret key k and database DB.
- $T \longleftarrow Trpdr(k, w_i)$:Trpdr algorithm, U_i generates a token T when it wish to search for specific W_i(to find out their identifiers). The token is then sent to CS, which conducts the search. The primary objective of this algorithm is to prevent the unauthorized disclosure of information stored on CS, in the form of an encrypted list of keywords (SI). Consequently, CS responds to user requests (T) in a secure manner.
- $d \longleftarrow Search(SI, T)$: A deterministic algorithm is executed by CS to search for data d (A set of identifiers)in the secure index SI, using a trapdoor T. In the case where d is encrypted, a resolve algorithm will be required.

- $R \longleftarrow Resolve(k,d)$: The DW executes this algorithm to recover the identifiers of keyword. The algorithm takes k and d as inputs and produces the final result R as output

B. The Advanced Encryption Standard (AES)

AES is an extensively used symmetric-key encryption algorithm that is designed for security and efficiency. Its purpose is to protect sensitive data and information during transmission and storage. AES was developed to replace the outdated and insecure Data Encryption Standard (DES). AES is a block cipher algorithm that uses a combination of substitution and permutation techniques to encrypt and decrypt data. It operates with a fixed plaintext block size of 128 bits (16 bytes), represented as a 4x4 matrix. The number of rounds in AES varies based on the size of the key, which can be either 128, 192, or 256 bits. The number of rounds for a 128-bit key is 10, for a 192-bit key it is 12, and for a 256-bit key it is 14. AES is widely recognized as one of the most secure encryption standards and is commonly employed in various applications, including securing online communications, financial transactions, and government/military communications [8, 9].

III. RELATED WORKS

In 2000, Song et al., they introduced a novel definition for SSE and presented various efficient techniques to implement SSE [10]. These mechanisms enabled Data Owners to securely store their data on an untrusted server, while ensuring that the server cannot retrieve the client's data. Since then, numerous other studies have emerged in this field. While most SSE studies have successfully adhered to the fundamental principles of SSE, practical experiments on large databases have revealed their poor performance, which degrades as the database size increases. This is primarily due to the bottleneck problem that they frequently encounter [11]. Studies in the literature have found that the bottleneck in these schemes was not due to encryption but rather to lower-level memory access issues, particularly poor locality. The known constructions can be broadly classified into two categories. The first approach is characterized by linear space and constant read efficiency but poor locality in [11,12]. Involves allocating an array of size N, where N elements of the database are uniformly mapped into the array. To retrieve a list of documents containing a specific keyword, each document identifier is stored in the array along with a pointer to the next document identifier in the list. Furthermore, the server needs to access random locations in the array with the number of identifiers that the word appears in. However, this approach is inefficient due to poor performance resulting from accessing a large number of different locations. The second approach achieves optimal read efficiency and locality but requires substantial space overhead [13–16]. The strategy behind this approach involves allocating a sufficiently large array and uniformly mapping the list of word identifiers into a contiguous interval in the array based on the length of word identifiers, without any overlaps among different lists. To efficiently retrieve a list for a given keyword, the server only needs to access a single random location and read all consecutive identifier entries, resulting in optimal read efficiency and locality. However, the locations of the lists in the array expose information about the structure of the underlying database. Therefore, padding is necessary to conceal information about the lengths of the lists, which leads to a polynomial space overhead. Therefore, developing a scheme with the best possible locality, storage, and read efficiency is a challenging task. In fact, David Cash and Stefano Tessaro demonstrated in 2014 [4] that it is impossible to achieve optimal performance in all three criteria simultaneously. They also established a lower bound on the tradeoff among these criteriaIn addition to their improvement on the locality by creating a scheme with logarithmic locality (log N), their scheme's storage space of O(N log N) was not optimal. In 2016, Gilad Asharov et al. [**?**] presented their third scheme, which updated David Cash and Stefano Tessaro's scheme to achieve O(1) locality with the same storage space. In 2017, Demertzis and Papamanthou [6] developed two schemes, the first of which had optimal locality and $O(NS_i)$ space, where S_i is the number of levels used to store data. However, this scheme had a minor impact on read efficiency and still required a large amount of storage space. The second scheme achieved a tunable locality, which could be selected as a parameter by the DW during the setup phase, while working within the same storage space as the first scheme. In 2021, Asharov et al. [7] made significant progress by developing two general frameworks - the pad-and-split framework and the statistical-independence framework - which strengthened the lower bound established by Cash and Tessaro. Throughout the recent period spanning 2021 to 2023, a multitude of research studies have surfaced across diverse domains within SSE, presenting numerous benefits. Nevertheless, all of them continue to exhibit a notable deficiency in terms of good locality such as [17–19].

IV. PROPOSED SCHEME

This section presents a proposed scheme that focuses on creating an inverted index of the database and modifying the mechanism for storing it to increase search speed. This modification involves converting the identifiers corresponding to each word (as stored in the traditional inverted index) into a single value for accessing the identifiers later. The proposed scheme is intended to be managed by three major components: the DW, CS, and t U_i. It consists of five distinct phases:

1. Key generation phase: The owner of the data generates a secret key.

2. Setup and Secure phase: The data owner creates a secure index and uploads it to the cloud server.
3. Token generation phase: When a user wants to search for a specific word, they create a token and send it to the cloud server for the search.
4. Secure search phase: The cloud server receives the token, searches for the required data in the secure index, and sends the result to the user.
5. Resolve phase: The user performs final manipulations on the result received from the cloud server to obtain the identifiers associated with the word.

CONSTRUCTION 1. (String based scheme)
Let $DB = \{DB(w_1), ..., DB(w_w)\}$
$M = \{(w_i), ..., (w_w)\}$;
For $w_i \in M$ $letDB(w_i) = \{(id_i), ..., (id_{nw})\}$ and n_{db} is total of identifiers DB Key generation phase $k\&k_v \leftarrow Gen(1^\lambda)$:
1. Input Security parameter 1^λ
2. Output k and $k_v, wherek\&k_v \in Z$
3. Compute both keys based on PRF

Setup and Secure phase $SI \leftarrow Enc(k, DB)$: 1. Input k and DB
2.Output $SI = H_T$
3.. Initialize empty H_T
4.For every $w_i \in M$
compute $la = PRF_k(1 \parallel w_i)$ and
$k_e = PRF_k(2 \parallel w_i)$
$Str =$ "", $i = 1$
For from i to n_{db}
if i in $DB(w_i)$
Add 1 to Str
Else
Add 0 to Str
$i = i + 1$
$\check{S}tr = Enc_{ke}(Str)$ by AES256
Add $(la, \check{S}tr)$ toH_T
5.uploading H_T to CS

Token generator phase $\check{T} \leftarrow Trqdr(k_v, k, w_i)$:
1. Input$k_v, kandw_i$
2.Output$\check{T}$
3.Compute $T = PRF_k(1 \parallel w_i) = la$
4.$\check{T} = Enc_T$
5.Send$\check{T}$ to CS

Secure Search phase $E\_\check{S}tr \leftarrow Search(k_v, \check{T}, SI)$
1.Input $k_v, \check{T}, SI$
2. Output $E\_\check{S}tr$
3.$T = Dec_v(\check{T}) = la$
4.$\check{S}tr = Get(la)$
5.Send $E\_\check{S}tr$ to U_i

Resolve phase $L\_ids \leftarrow Resolve(k_v, k, E\_\check{S}tr)$
1. Input $k_v, kandE\_\check{S}tr$
2.Output L_ids = identifiers 3.Restore $PRF_k(2 \parallel w_i) = k_e$
4.$\check{S}tr = Get(la)$
5.Initialize empty L_ids and i=1
6. For from i to length Str
if Str[i]=1
Add i to L_ids
i=i+1

V. THE PROPOSED SCHEME WITH MORE DETAIL

A. Key generation phase

In this phase, DW generates a secret key k using a Pseudo-Random Function (PRF), which can be defined as follows:
A PRF function $F : \{0,1\}^* x \{0,1\}^* \rightarrow \{0,1\}^*$ is used to create the key. It takes two inputs k and an input and produces an output that cannot be distinguished from a truly random function, except with negligible probability in 1^λ , denoted as $negl(1^\lambda)$ [20, 21]
The key k is used in the setup and secure phase to encrypt the data $SI \leftarrow Enc(k, DB)$, and it is sent to all trusted users. In the subsequent phases, such as the token generator phase,$\check{T} \leftarrow Trpdr(k_v, k, w_i)$, and the resolve phase, $L\_ids \leftarrow Resolve(k_v, k, E\_\check{S}tr)$, U_i must use k.
Furthermore, another key called k_v is created using the same method, and it is used to secure the trapdoor T and encrypted string $\check{S}tr$ exchanged between the major components CS andU_i
.

B. Setup and secure phase

The current phase involves the DW configuring a secure index called SI, which will be uploaded to CS upon completion. The main inputs for this process are k and DB, and the output is SI. The following steps provide an explanation of how this mechanism works:

Create an empty hash table H_T that will serve as a foundation for storing secure parameters and subsequently used as SI for retrieving user's requests.

For each keyword w_i extracted from M, where $w_i \varepsilon M$, performed a series of consecutive steps outlined below:

1. Using PRF to derivation of two keys from w_i. The first one la is computed , which takes 1 and w_i as its inputs as follows $la = PRF_K(1 \parallel w_i)$ it is used as a label later to store $\check{S}tr$ in H_T. The second keyk_e creates in the same way$k_e = PRF_k(2 \parallel w_i)$), it is used to encrypted Str.

2. Initialize the initial values as empty string Str and i=0

3. The advantage of Str is its usefulness in expressing the presence the identifiers by which the word appears. This string has a length equivalent to the database identifiers n_{db} and comprises two values: 0 and 1. It is created by adding 1 to the rank corresponding to the identifier number in Str. If not, then 0 is added. For example, if $DB(w_i) = 1,3,5,8$ and $n_{db} = 10$, then the binary string representation would be $Str = ''1010100100''$.

4. Encrypt Str by k_e and $\check{S}tr$ represents an encrypted string.

5. Add the $(la, \check{S}tr)$ pair as a $(key, value)$ into the H_T uploadingH_T to the cloud server

C. Trapdoor phase

In the present phase, U_i conducts a search for a specific word w_i. Trpdr is responsible for processing w_i, k_v and k as input parameters, resulting in the production of $\check{T}$ as output. Which will compute T and subsequently secured usingk_v, with the following formula:$\check{T} = Enc_{kv}(T)$), which enables more secure transfer to CS.

D. Secure search phase

After receiving a user's query $\check{T}$,and decryption $\check{T}$, $T = Dec_{kv}(\check{T}) = la$, CS responds by returning $\check{S}tr$ stored in SI by $\check{S}tr = Get(la)$ It is encrypted and sent to U_i securely $E\_\check{S}tr = Enc_{kv}(\check{S}tr)$

E. Resolve phase

At this phase, the user receives $E\_\check{S}tr$ from CS and follows the subsequent steps to obtain the identifiers.

1. Decrypt $E\_\check{S}tr$ and then decrypt $\check{S}tr$ to get Str

2. Initialize empty L_ids it is a list used to store identifiers after obtaining them and i=1. 3. For from i to length Str
If $Str[i] = 1$
Add i toL_{ids}
$i = i + 1$

VI. SECURITY ANALYSIS

In the literature review, there are various syntaxes for SSE schemes, which can be categorized into two types based on how the cloud server and DW interact during each query for data. The first type is a single-round interaction, where the server decrypts the data and sends the result to the DW, thus learning the output in the process. In contrast, the second type utilizes multiple rounds of interaction, where the cloud server does not learn any information about the output [**?**].

The proposed scheme prioritizes the security of data exchanged over the communication channel between cloud server and DW, and therefore, we have opted to use the second type of SSE syntax that employs multiple rounds of interaction to ensure that the cloud server does not gain access to any information about the output.

A. Server information leakage

The concept of leakage functions L refers to the amount of information that can be learned by the cloud server about the stored data. These functions can be classified into three types: L_{max} and L_{min} , which are associated with single-round interaction SSE, and L_{size} , which is used for multiple rounds of interaction. All three types take M and DB as input parameters [4, 7].

$I_{max}(M, DB)$: the output is $(N, \{DB(w_i)\}_{w_i \varepsilon M}, W, n_{db}, n_w$ $, Max(DB(w_i))_{w_i \varepsilon M})$This function represents the maximum possible amount of leakage.

$I_{min}(M, DB)$: output $(N, \{DB(w_i)\}_{w_i \varepsilon M})$This type of leakage is considered somewhat acceptable. It is important to note that the cloud server must know N in all cases. During a single-round interaction, the server can view the query results because it is capable of decrypting the output.

$I_{size}(M, DB)$: output of function is $(N, \{|DB(w_i)|\}_{w_i \varepsilon M})$

In multi interaction rounds, the cloud server responds to the user's request without decrypting the results and will only learn the size of the results. Our work falls into this type of leakage because we employed multiple rounds of interaction. However, it's important to note that the encrypted values of the Str parameter stored in the cloud server are of equal length, but they do not represent the actual identifiers themselves. Rather, they are strings that indicate the presence or absence of a particular identifier. Consequently, the uniform output size of the cloud server does not disclose any information

about n_w.

B. Resisting attacks

In this section, we explored the robustness of our scheme and the most well-known forms of attacks on SSE.

1) KGA (Known-Keyword Attack)

KGA attack is a form of attack that targets searchable symmetric encryption (SSE) systems. In a KGA attack, the CS tries to infer the keyword that a user is searching for, and leverages this information to retrieve the encrypted data associated with that keyword. This type of attack can undermine the privacy and confidentiality of the stored data, and can potentially allow CS to launch additional attacks on the system. KGA attacks are among the most prevalent types of attacks that are launched against SSE schemes [22]. Our scheme employs multiple precautionary measures to resist attack, including encrypting the words and ensuring the key is kept secret and secure. As a result, we are confident that our work can effectively resist KGA attacks.

2) Frequency analysis attack

A frequency analysis attack is a type of cryptographic CS used to break a cipher by analyzing the frequency of occurrence of letters or symbols in the encrypted data. Therefore, the method exploits the frequency of the encrypted data that is uploaded to SI, which can be represented as either term frequency (TF) or term frequency-inverse document frequency (TF-IDF) [23]. TF is the measure of the number of times a term w_i appears in a document . On the other hand, TF-IDF is the product of TF and IDF values. IDF is a measure of how important w_ii is to a collection of documents. It is calculated as the logarithm of the ratio of the total number of documents in the collection $n_{db}$to the number of documents that contain w_i. CS potentially carry out this attack and determine the keyword being searched for if they have access to this important information. However, our work is designed to protect against such attacks as the values stored in CS are encrypted and do not reflect the original identifiers directly. Instead, they are represented by a text that helps the user U_i to access the identifiers later. Therefore, we can confidently state that our work is secure against this type of attack.

3) Inverted Keyword Knowledge IKK attack

IKK attack is used to determine the plaintext words that were used by the user U_i to search for trapdoors [24]. This attack relies on the disclosure of partially known information, specifically the leaking of the access pattern information. The access pattern information is defined as the result of the search for T in SI by CS. To provide an example, consider a database that contains information about networks, and a user submits three queries as trapdoors: T_1, T_2 and T_3 , which correspond to the words "communications," "computers," and "link," respectively. After the communication between the user U_i and CS is completed, the CS obtains the results, which are the set of identifiers corresponding to the trapdoors. The CS can then calculate the probability of any two of these keywords appearing together in any document by analyzing the number of documents that are common between the corresponding trapdoors. By continuing the search and gradually revealing the access pattern, CS can obtain more information about the probabilities of the keywords corresponding to the trapdoors [24]. Eventually, through this process, the server may be able to determine the keywords that correspond to the trapdoors. However, our scheme is designed to resist this type of attack. Specifically, the search results obtained by the CS are encrypted, and the access pattern does not leak any important information. Therefore, the CS will not be able to access the identifiers that correspond to the trapdoors, which ensures the security of our scheme against the IKK attack.

4) Man-in-the-middle attack MITM

This type of attack occurs when the communication channel between the user U_i and CS is not secure, as an attacker can impersonate one of the parties [25]. To prevent this type of attack, we have implemented several security measures in our work. Firstly, we secure the communication channel by encrypting the exchanged data (T and $\check{S}tr$) between the CS and U_i using k_v key. Additionally, we ensure mutual authentication between the two parties using the same key. These steps are taken during each the Token generator phase, secure search phase, and resolve phase as the following:

CONSTRUCTION 2. In token generator phas:
U_i selects a random identifier ID_U.
$H_{ID_U} = H_{kv}(ID_U)$
Send $(H_{ID_U} \parallel ID_U \parallel \check{T})$ to CS
In Secure Search Phase:
CS selects a random identifierID_{CS}.
if $H_{ID_U} = vrfy_{kv}(ID_v) \rightarrow U_i$ authentication
Before send $E\_\check{S}tr$
$H_{ID_{CS}} = H_{kv}(ID_{CS})$
Send $(H_{ID_U} \parallel ID_U \parallel E\_\check{S}tr)$ to U_i
In Resolve Phase:
if $H_{ID_{CS}} = vrfy_{kv}(ID_{CS}) \rightarrow CS$ authentication

VII. Experimental Results

In this section, the performance of the proposed scheme was evaluated using a real-data collection of Wikipedia articles. A database of 2250 files was selected, with $n_{db} = 2250$ and 112300 words, W=112300. The study was conducted on a 64-bit Windows 10 PC, which was equipped with an Intel Core i5 CPU clocked at 2.6 GHz and 8 GB of RAM. Python was chosen as the programming language for its numerous functionalities.

A. Comparison with previous schemes

As an initial step to ensure the effectiveness of the approach, several prior schemes [4–6, 26], were implemented. In the second phase, an experiment was conducted on the same DB and w_i, utilizing the AES256 encryption algorithm to evaluate the search time required to retrieve the identifiers associated with the word. This experiment involved the proposed approach as well as four other previously implemented schemes. The word that occurs in the largest number of files, $n_w = 1933$, was selected for the evaluation. The successful outcomes of the efforts to enhance the pace of information retrieval are demonstrated in **??**.

To ensure a fair comparison with other studies, the duration of the resolve phase was included in addition to the research time, which was not incorporated in previous schemes.

In the second experiment, a word that appeared only in two files, $n_w = 2$, was focused on. The outcomes demonstrated that search speed was remarkably high across all schemes, including the proposed method. This outcome was anticipated as the number of identifiers was small. However, the experiment verified that the adjustments made to the encrypted inverted index storage mechanism did not result in negative impacts when the number of identifiers was limited, as shown in **??**.

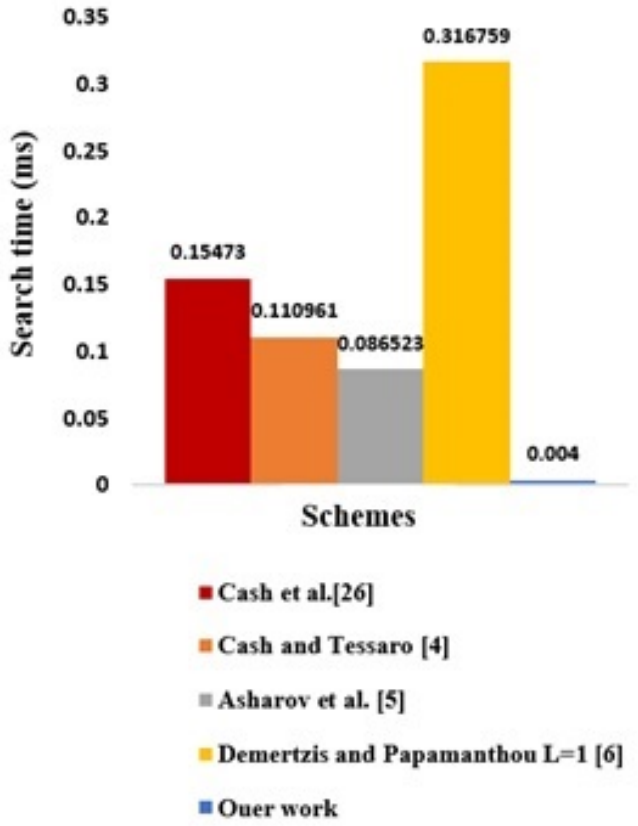

Fig. 1. Comparison the time it takes to search for a word with $n_w = 1933$ in proposed scheme with the time required by previous schemes.

B. Discussion

In this discussion, we will delve into the factors that have contributed to the success of the proposed scheme and how it differentiates itself from previous schemes that underwent evaluation alongside it. It is clear from the practical experiments that were conducted, the time taken to perform a search operation is predominantly influenced by several factors, including the locality (i.e., the frequency of accessing various memory locations within CS), the structure of the encrypted index stored on CS, and the number of decryption operations required to retrieve the relevant identifiers. Certain schemes can be significantly impacted by poor locality, as in reference [26] in the experiment, the first word retrieved necessitated the cloud server to move to 1933 different positions, resulting in a potential delay in search time as shown in **??**. The search process may experience delays due to the structure of the index stored on the cloud server and the frequency of decryption required, even if the locality is relatively good as in [4–6].

Regarding the suggested scheme, it exhibits optimal locality O(1), resulting in the cloud server moving only once instead of 1933 positions during the search process. Additionally, decryption is only required once, which significantly improves the speed of access to identifiers. These factors, in the proposed scheme, have contributed to its success and set it apart from previous schemes.

VIII. Conclusions

This paper addresses the issue of suboptimal performance in large databases caused by cloud server accessing data from multiple positions during the search process to respond to a single query from the user. The primary focus of this paper

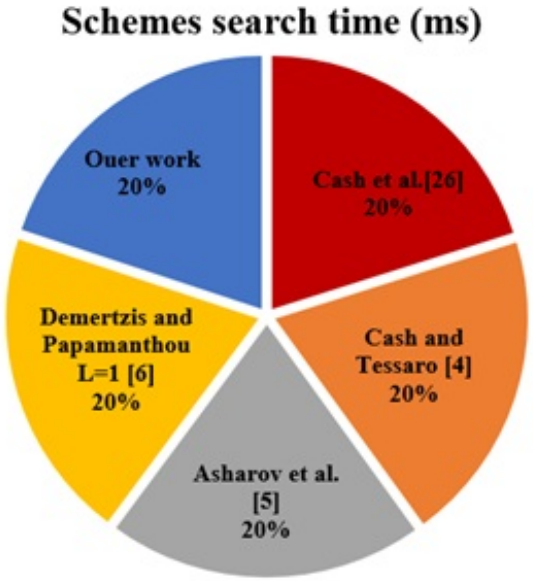

Fig. 2. Comparison the time it takes to search for a word with $n_w = 2$ in proposed scheme with the time required by previous schemes.

is on the issue of poor locality, and a solution is presented to improve the performance by modifying the inverted index storage mechanisms. The proposed scheme ensures optimal locality O(1), which enhances the overall performance of searchable symmetric encryption. Additionally, the proposed scheme boasts robust security measures, significantly reducing information leakage to the server and making it highly resistant to most known SSE attacks. Furthermore, experiments were conducted using real-world data to demonstrate the practical efficiency and accuracy of the proposed scheme.

Conflict of Interest

The authors have no conflict of relevant interest to this article.

References

[1] H. Akbar, M. Zubair, and M. S. Malik, "The security issues and challenges in cloud computing," *International Journal for Electronic Crime Investigation*, vol. 7, no. 1, pp. 13–32, 2023.

[2] A. Ahmed, S. Kumar, A. A. Shah, and A. Bhutto, "Cloud computing security issues and challenges," *Tropical Scientific Journal*, vol. 2, p. 1–8, Jan. 2023.

Character	Description
w_i	Word
W	Number of words
M	Words in DB $M = w_i, ..., w_w$
id	Identifier
n_{db}	Total of identifiers DB
n_w	Total of identifiers w_i
N	$\sum_{i=1}^{w} \lvert DB(w_i)\rvert$ where $DB(w_i) = \{id_i, ..., id_{n_w}\}$
PRF	Pseudo-random function
H_T	A hash table is a type of data structure used for storing and retrieving data. It comprises a pair of algorithms, namely "Add" and "Get," [7] which enable efficient and fast access to stored information.
Add	Algorithm adds pairs of (key , value) to H_T
Get	value=Get(key)
Str	String
$\check{S}tr$	Encrypted string
la	Label is used to store and retrieve $\hat{S}$ in H_T , $Add(la, \check{S}tr), \hat{S} = Get(la)$
Enc	Function to encryption Str
Dec	Function to decryption $\check{S}tr$
la	Derivative key to create la
k_e	Derivative key to encrypted and decrypted
$\check{T}$	Encryption T

[3] G. S. Poh, J.-J. Chin, W.-C. Yau, K.-K. R. Choo, and M. S. Mohamad, "Searchable symmetric encryption: designs and challenges," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 1–37, 2017.

[4] D. Cash and S. Tessaro, "The locality of searchable symmetric encryption," in *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33*, pp. 351–368, Springer, 2014.

[5] G. Asharov, M. Naor, G. Segev, and I. Shahaf, "Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pp. 1101–1114, 2016.

[6] I. Demertzis and C. Papamanthou, "Fast searchable encryption with tunable locality," in *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 1053–1067, 2017.

[7] G. Asharov, G. Segev, and I. Shahaf, "Tight tradeoffs in searchable symmetric encryption," *Journal of Cryptology*, vol. 34, pp. 1–37, 2021.

[8] A. M. Abdullah *et al.*, "Advanced encryption standard (aes) algorithm to encrypt and decrypt data," *Cryptography and Network Security*, vol. 16, no. 1, p. 11, 2017.

[9] Y. Alemami, M. A. Mohamed, and S. Atiewi, "Advanced approach for encryption using advanced encryption standard with chaotic map," *Int. J. Electr. Comput. Eng*, vol. 13, pp. 1708–1723, 2023.

[10] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*, pp. 44–55, IEEE, 2000.

[11] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pp. 353–373, Springer, 2013.

[12] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 965–976, 2012.

[13] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pp. 577–594, Springer, 2010.

[14] P. Van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker, "Computationally efficient searchable symmetric encryption," in *Secure Data Management: 7th VLDB Workshop, SDM 2010, Singapore, September 17, 2010. Proceedings 7*, pp. 87–100, Springer, 2010.

[15] K. Kurosawa and Y. Ohtaki, "How to update documents verifiably in searchable symmetric encryption," in *Cryptology and Network Security: 12th International Conference, CANS 2013, Paraty, Brazil, November 20-22. 2013. Proceedings 12*, pp. 309–328, Springer, 2013.

[16] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*, pp. 258–274, Springer, 2013.

[17] H. M. Mohammed and A. I. Abdulsada, "Secure multi-keyword similarity search over encrypted data with security improvement.," *Iraqi Journal for Electrical & Electronic Engineering*, vol. 17, no. 2, 2021.

[18] H. M. Mohammed and A. I. Abdulsada, "Multi-keyword search over encrypted data with security proof," *Journal of Basrah Researches (Sciences)*, vol. 47, no. 1, 2021.

[19] C. Guo, W. Li, X. Tang, K.-K. R. Choo, and Y. Liu, "Forward private verifiable dynamic searchable symmetric encryption with efficient conjunctive query," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[20] J. Katz and Y. Lindell, "Introduction to modern cryptography crc press," 2020.

[21] Y. Watanabe, T. Nakai, K. Ohara, T. Nojima, Y. Liu, M. Iwamoto, and K. Ohta, "How to make a secure index for searchable symmetric encryption, revisited," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 105, no. 12, pp. 1559–1577, 2022.

[22] Y. Miao, Q. Tong, R. H. Deng, K.-K. R. Choo, X. Liu, and H. Li, "Verifiable searchable encryption framework against insider keyword-guessing attack in cloud storage," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 835–848, 2020.

[23] D. Siva Kumar and P. Santhi Thilagam, "Searchable encryption approaches: attacks and challenges," *Knowledge and Information Systems*, vol. 61, pp. 1179–1207, 2019.

[24] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 668–679, 2015.

[25] S. Gangan, "A review of man-in-the-middle attacks," *arXiv preprint arXiv:1504.02115*, 2015.

[26] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," *Cryptology ePrint Archive*, 2014.