Enhancing Searchable Symmetric Encryption Performance through Optimal Locality

Aya A. Alyousif[1*], Ali A. Yassin[2], Hussein M. Mohammed[3]
[1] Department of Medical Instrumentation Engineering Techniques, Shatt Al-Arab University College, Basra, Iraq
[2] Department of Computer Science, Education College for Pure Sciences, University of Basrah, Basrah, Iraq
[3] Directorate General of Education Basrah, Ministry of Education, Basrah, Iraq
Email: ayah.abdulhussain@sa-uc.edu.iq[1], ali.yassin@uobasrah.edu.iq[2] , hussain.mazin@sa-uc.edu.iq[3]
[*]Corrsponding author

Keywords: information retrieval, searchable symmetric encryption, inverted index, locality, cloud server

Received: March 17, 2024

Both individuals and institutions place great importance on maintaining the security and privacy of their data, when stored in the cloud server. To achieve this, they often turn to searchable symmetric encryption (SSE), which is considered a crucial technology for safeguarding user data. However, SSE has encountered some challenges, particularly in the case of large databases. One such issue is poor performance, which can be attributed to poor locality. This means that the cloud server must visit a large number of locations during the search process, resulting in slow retrieval times. The main problem however, is not just poor locality. In many cases, optimization methods intended to improve performance can actually lead to increased storage requirements for the encrypted index stored on the cloud server or reduced efficiency when reading data. These issues must be addressed in order for SSE to continue to be an effective tool for protecting sensitive information. In this paper, we introduce a secure and searchable scheme that effectively addresses the issues mentioned above, while also enhancing the performance of information retrieval through an improved encrypted inverted index storage mechanism. Our scheme achieves optimal locality at $O(1)$, and read efficiency at $O(1)$, thereby significantly increasing the speed of retrieval. Through experimentation with real-world data, we have demonstrated the practicality, accuracy, and security of our approach, making it a reliable solution for secure and efficient information retrieval.

Povzetek: Predlagana je optimizirana metoda za izboljšanje učinkovitosti iskanja v šifriranih podatkih z uporabo izboljšanega obrnjenega indeksa, ki dosega optimalno lokalnost in povečuje hitrost ter varnost pridobivanja informacij.

1 Introduction

In the modern age, the need for data storage has greatly increased. With technology advancing rapidly, we are generating data faster than ever, and businesses, individuals, and organizations all need efficient ways to store, manage, and access this data. Consequently, cloud storage has become a popular choice, offering many benefits over traditional data storage methods[1][2].

Cloud storage refers to storing data on remote servers accessible via the internet from any connected device. This method allows data to be centrally stored and easily accessed from anywhere with an internet connection. Cloud storage is flexible, letting users adjust their storage needs up or down as required, without the limitations of physical storage devices. It is ideal for businesses and individuals needing to store large amounts of data securely. Cloud storage providers ensure high reliability by using redundant storage systems and multiple data centers in different locations, making data always accessible, even during hardware failures or natural disasters. This reliability is crucial for businesses relying on data for critical operations, ensuring their data is secure and available when needed.

Despite the numerical advantages of cloud storage, it is not without challenges, and security is the biggest challenge for cloud storage [3]. To ensure the security of data stored on cloud servers, various methods need to be employed, including access control, network security, and encryption [4]. Access control is a mechanism that restricts access to data based on user identity, role, or authorization. Network security involves securing the network infrastructure used for data transmission. Encryption, on the other hand, is the process of transforming data into code to prevent unauthorized access. Encryption can be applied both in transit and at rest, thus ensuring that data remains secure during transmission and storage.

To secure data stored in the cloud, various encryption techniques are used to prevent unauthorized access. Symmetric encryption is widely used and involves a single key for both encrypting and decrypting data. In contrast, asymmetric encryption uses different keys for encryption and decryption, offering more security but at a slower speed. Hashing is another method, commonly used for securing passwords, which converts data into a fixed-size string that cannot be reversed. A newer technique, Searchable Symmetric Encryption (SSE) [5], allows users to search encrypted data without exposing it. SSE comes in two types: deterministic, which offers consistent results but less security, and probabilistic, which provides better security but less predictability.

Table 1: List of symbols

Character	Description
W	Word
nw	Number of W
M	Words in $DB,\ M = \{W_1, \ldots, W_{nw}\}$
id	Identifier
Ndb	Total of identifiers DB
n	Total of identifiers W
N	$\sum_{i=1}^{nw} \lvert db(W) \rvert$ where $db(W) = \{id_1, .., id_n\}$
c	Counter
H_T	A hash table is a data structure that allows efficient storage and retrieval of key-value pairs. It comprises a pair of algorithms, are "Add" and "Get"[11].
Add	Algorithm adds pairs of $(key, value)$ to H_T
Get	value=Get(key)
S_t	String
$\check{S}_t$	Encrypted string
sk_e	Derivative key used for encryption and decryption of S_t
La	Label is used to store and retrieve $\check{S}t$ in HT, $Add(La, \check{S}_t)$, $\check{S}_t = Get(La)$
Enc	Function to encryption S_t
Dec	Function to decryption $\check{S}_t$
L_{id}	List to store identifiers

These encryption methods are crucial for protecting cloud data and must be carefully implemented to prevent data breaches or attacks.

Searchable symmetric encryption (SSE) presents several challenges[6]. The process, which involves searching encrypted data, requires creating and maintaining an index for each keyword, making it complex. Moreover, there are security risks, such as the possible leakage of sensitive data. A recent challenge in SSE is a significant drop in performance and retrieval efficiency when dealing with large databases [7]. Researchers have found that this issue is not due to flaws in the encryption itself but is related to how the secure index is stored in memory. During a search, the index may cause the cloud server to perform many continuous memory transitions, known as "poor locality," " [[7], [8], [9], [10], [11]] which can slow down the retrieval process and degrade SSE performance. While some researchers are working to improve locality to boost performance, this can lead to increased storage requirements for the encrypted index on the cloud server or reduce the efficiency of data reading. The contributions of our approach can be summarized as follows:

Firstly, our scheme significantly improves the performance of information retrieval for all databases, regardless of their size, by enhancing locality. Secondly, our scheme achieves an optimal locality of $O(1)$, meaning that the cloud server only needs to access one memory location during each search operation, as opposed to many locations. Thirdly, our proposed scheme is highly secure, as the server searches for the required data and sends it to the data owner without decrypting it, thereby enhancing resistance to various attacks that symmetric searchable encryption is vulnerable to. Fourthly, our scheme has better reading efficiency $O(1)$, as the cloud server only responds with the requested data when the user queries it. Finally, our scheme has no significant negative impact on the storage of the encrypted index in the cloud server.

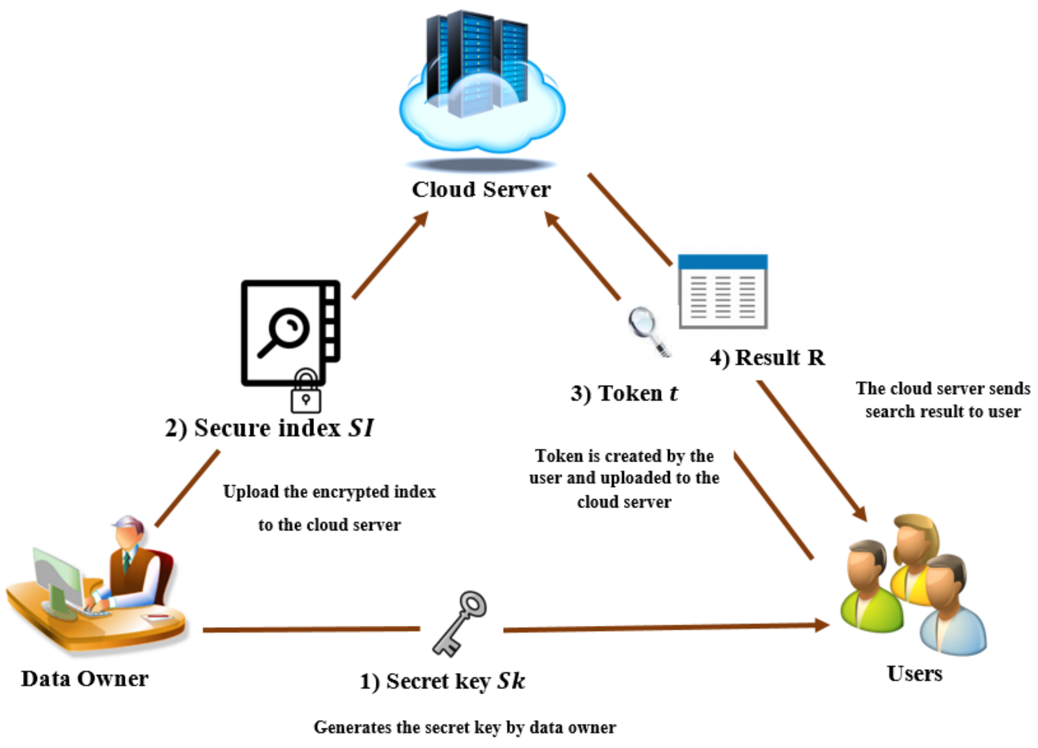
Cloud Server
2) Secure index SI
Upload the encrypted index to the cloud server
3) Token t
Token is created by the user and uploaded to the cloud server
4) Result R
The cloud server sends search result to user
Data Owner
1) Secret key Sk
Generates the secret key by data owner
Users

Figure 1: Searchable symmetric encryption

Table 2: Searchable symmetric encryption algorithm

Algorithm	Description
Key generation phase: $\boldsymbol{sk \leftarrow Gen_{sk}(1^{\lambda})}$	The key generation algorithm takes the security parameter1^{λ} as input and generates the secret key as its output.
Constructing secure index: $\boldsymbol{SI \leftarrow Enc\_DB(sk, DB)}$	The secure index SI is created in this algorithm by taking the secret key sk and the database DB as input.
Token generator: $\boldsymbol{t \leftarrow Trpdr\_W(sk, W)}$	The token t in this algorithm is created by the user to search for a specific word W or data.
Search: $\boldsymbol{R \leftarrow Search\_t(t, SI)}$	In this algorithm, the cloud server searches for the required word in SI and returns the result R to the user. If the result is encrypted, the user will need to use $Find\_ids$ algorithm.
Find identifiers: $\boldsymbol{IDS \leftarrow Find\_ids(sk, R)}$	The user employs this algorithm to retrieve the final outcome, which comprises word identifiers IDS after carrying out any essential processing and decrypting R

2 Searchable symmetric encryption (SSE)

SSE is a technology designed to enable searching over encrypted data while maintaining its confidentiality [[12], [13], [14]]. It involves three key components: the data owner DW, the cloud server CS, and the users. The process of SSE consists of several steps. Initially, the data owner selects a secret key for encryption and decryption. Next, they construct a secure index based on the words in their database. This index is then encrypted with the secret key chosen earlier. When a user wishes to search for data, they generate an encrypted search query known as a token. This token is encrypted using the same secret key that was used to encrypt the index and is sent to the cloud server. Upon receiving the token, the cloud server uses the encrypted index to search for the requested data[15]. The cloud server either decrypts the data and sends it to the user or sends it encrypted so that the user can decrypt it later. The Table 2. summarizes the steps involved in Searchable Symmetric Encryption algorithms:

Table 3: Comparison with previous schemes based on storage, locality, and read efficiency

Related works	Storage	Locality	Read efficiency
Curtmola et al. [14]	$O(N)$	$O(n)$	$O(1)$
Kamara et al. [21]	$O(N)$	$O(n)$	$O(1)$
David cash et al. [17]	$O(N)$	$O(n)$	$O(1)$
Chase and Kamara [18]	$O((\text{Max}\lvert db(W)\rvert)Ndb)$	$O(1)$	$O(1)$
P. van Liesdonk et al.[19]	$O(nw\ Ndb)$	$O(n)$	$O(1)$
Kamara and Papamanthou [22]	$O(nw\ Ndb)$	$O(n\ log\ Ndb)$	$O(Ndb\ logNdb)$
David cash et al. [8]	$O(N\ log\ N)$	$O(log\ N)$	$O(1)$
Asharov et al. (Scheme 3) [9]	$O(N\ log\ N)$	$O(1)$	$O(1)$
Demertzis and Papamanthou [10]	$O\ (N\ S)$	$O(L_d)$ Where L_d is a tunable locality	$O(\frac{N^{\frac{1}{s}}}{L})$
Asharov et al. (Pad-and-split scheme) [11]	$O(N\ log\ N/\ log\ L)$	$O(L_d)$ Where L_d depends on the scheme in which it is implemented within its framework	$O(1)$
Alyousif et al.[23]	$N = \sum_{i=1}^{nw}(\sum_{j=1}^{nq} St)$	$O(nq)$	$O(1)$

	where nq is number of QR codes for the word and St is a string		
Our work	$O(N)$	$O(1)$	$O(1)$

3 Related works

In the year 2000, a novel technology emerged that facilitated the searching of encrypted data without the need for decryption [16]. The system was dubbed Searchable Symmetric Encryption and provided users with the ability to search for specific keywords within encrypted data while maintaining content security. SSE's inception marked a significant milestone in the field of data security and privacy. Following the introduction of this new technology, extensive research was conducted in various areas, including performance optimization. Studies have shown that the reduction in performance is not due to the technology itself, but rather to the memory positions that the server accesses while processing user requests. As the encrypted index size grows, the number of positions accessed also increases, resulting in a slower response time [17]. This case is called poor locality. Known constructions can be classified into two approaches. The first approach has linear space and constant read efficiency, but its poor locality is highlighted in references [14] and [17]. This scheme involves allocating an array of size N and uniformly mapping N elements from the DB into it. To retrieve a list of identifiers that contain a specific W, the approach stores each identifier in the array alongside a pointer to the next identifier in the list. Unfortunately, this approach requires the cloud server to access random positions in the array, with the number of identifiers associated with the word, which results in inefficiency due to the need to move to a large number of different positions. The second approach has excellent read efficiency and locality, but it comes at the cost of a significant amount of extra space [[18], [19], [20], [21]]. The basic idea behind this approach is to allocate a sufficiently large array A and then map each list of W identifiers uniformly into a contiguous interval in A, based on the length of the W identifiers. There should be no overlaps among different lists. To retrieve a list for a given W efficiently, the cloud server only needs to access one random position and read all consecutive identifier entries, which leads to optimal read efficiency and locality.

However, the positions of the lists in the array reveal information about the structure of the underlying DB. To hide this information, padding must be applied, resulting in a polynomial increase in space usage. We have to highlight that there is often a problem with storage capacity, which is often large due to locality optimization, or bad locality itself which can have a detrimental effect on cloud server response time. And sometimes there is a negative impact on the efficiency of reading the data as well. It is difficult, if not impossible, to construct a construction that is ideally locality with limited storage space without compromising its data read efficiency. This issue was discussed by Cash and Tessaro in 2014[8], where they also determined the minimum tradeoff required between these three criteria. Also, Cash and Tessaro developed a new construction that enhances locality to $O(log\ N)$ with storage capacity $O(N\ log\ N)$. In 2016, Gilad Asharov et al. improved construction locality for Cash and Tessaro locality, achieving locality of $O(1)$ [9] while maintaining the same storage capacity. Demertzis and Papamanthou [10] developed two constructions in 2017. The first construction offered optimal locality and required $O(N\ S)$ space, where S represents the number of levels employed to store data. However, this construction resulted in a slight reduction in read efficiency, and the storage space needed was still significant. The second construction, which operated within the same storage space as the first, allowed for tunable locality, enabling the DW to select a parameter through which to create their index. In 2021, Asharov et al. achieved remarkable progress by introducing two frameworks [11]: pad-and-split and statistical-independence.

The last work that we will talk about is one of our previous works [23] that aims to improve locality using QR code technology. The work achieved good results compared to previous works, but the locality was not ideal O(1) rather, it depends on the number of QR codes for the word. The following table summarizes the most important previous works in terms of the three main important characteristics: locality, storage efficiency, and reading efficiency.

4 Proposed scheme

In this section, we will explain our scheme in detail. First, the data owner generates a secret key, sk, using a

```
CONSTRUCTION.
 Let $= \{db(W_1), \ldots, db(W_{nw})\}$ ,
; For $W \in M$ let $db(W)$ =$M = \{W_1, \ldots, W_{nw}\}$
 $\{id_1, .., id_n\}$ and $Ndb$ is total of identifiers $DB$
:$\boldsymbol{sk \leftarrow Gen_{sk}(1^{\lambda})}$
 Compute $sk$ with $PRF$
:$\boldsymbol{H\_T \leftarrow Enc\_DB(sk, DB)}$
1. Initialize empty $H\_T$
2. For every $W \in M$
    Sort $db(W)$
    $S_t$ = " ", i=1
    For from $i$ to $Ndb$
        If $i$ mod 2 is not equal 0
            If $i$ and $i+1$ in $db(W)$
                Add "A" to $S_t$
            Else if $i$ and $i+1$ not in $db(W)$
                Add "D" to $S_t$
            Else if $i$ in $db(W)$ and
            $i+1$ not in $db(W)$
                Add "B" to $S_t$
            Else if $i$ not in $db(W)$ and
            $i+1$ in $db(W)$
                Add "C" to $S_t$
    $sk_e = PRF_{sk}(2 \parallel w)$
    $\check{S}_t = Enc_{sk_e}(S_t)$ by AES256
    Compute $La = PRF_{sk}(1 \parallel W)$
    Add $(La, \check{S}_t)$ to $H\_T$
```

Pseudo-Random Function (PRF). A PRF is a deterministic algorithm that produces outputs that seem random, despite being generated from a specific input. PRFs are commonly used in cryptography to improve security, as their output is difficult to distinguish from true randomness, even against powerful adversaries[[24], [25],[26]]. The secret key, sk, will be used for both encryption and decryption. Below, we present the complete construction of our scheme.

The next step involves the data owner DW creating a secure index SI for the database DB. In our approach, SI is equivalent to H_T,, which is based on the words M in the database and their corresponding identifiers $db(W) = \{id_1, .., id_n\}$. The DW then arranges the identifiers $db(W)$ for each word W in the group M in ascending order and prepares an empty string S_t = "" for the next step.

Next, we examine all identifiers in odd positions within the database. For each of these odd-positioned identifiers, we check the state of the corresponding word as well as the adjacent even-positioned identifier, recording the results in S_t. There are four possible scenarios, labeled A, B, C, and D, depending on whether the word appears in the odd or even positions.

1. Case A: If the word appears in the identifiers that is located in the odd and even positions together.

```
:$\boldsymbol{t \leftarrow Trpdr\_W(sk, W)}$
1. Input $sk$ and $W$
2. Compute $t = PRF_{sk}(1 \parallel W) = La$
$\boldsymbol{\check{S}_t \leftarrow Search\_t(t, H\_T)}$:
$\check{S}_t = Get\ (La)$
:$\boldsymbol{L_{id} \leftarrow Find\_ids(sk, \check{S}_t)}$
1. $sk_e = PRF_{sk}(2 \parallel W)$
2. $S_t = Dec_{sk_e}(\check{S}_t)$
3. Initialize $L_{id}$=[] list and $i = 1$
c=0
 For from $i = 1$ to $Ndb/2$
        if $S_t[i]$ equal "A"
                c=c+1, add c to $L_{id}$
                c=c+1, add c to $L_{id}$
        Else if $S_t[i]$ equal "D"
                c=c+2
        Else if $S_t[i]$ equal "B"
                c=c+1, add c to $L_{id}$
                c=c+1
        Else if $S_t[i]$ equal "C"
                c=c+1
                c=c+1, add c to $L_{id}$
```

2. Case D: The opposite of Case A is when the word does not appear in either identifier located in the odd and even positions.
3. Case B: If the word appears in the identifier located in an odd position but not in the identifier located in an even position.
4. Case C: Conversely, the opposite of Case B is when the word does not appear in the identifier located in an odd position but appears in the identifier located in an even position.

Once we have constructed S_t that indicates the presence or absence of the word in its identifiers, we will generate two keys. The first key $sk_e = PRF_{sk}(2 \parallel w)$ will be used to encrypt S_t, $\check{S}_t = Enc_{sk_e}(S_t)$, ensuring that it remains secure and confidential. The second key $La = PRF_{sk_e}(1 \parallel W)$ will serve as a label to identify $\check{S}_t$ when it is stored in the hash table.

After encrypting S_t with the first key, we store the encrypted result, $\check{S}_t$, in a hash table using La as a label. We then add $(La, \check{S}_t)$ to H_T, which allows us to easily retrieve $\check{S}_t$ when needed while keeping it secure from unauthorized access or tampering. Once H_T is constructed, the data owner DW can upload it to the cloud server CS. When a user wants to search for a word W, they generate a token t using the second key La, which was created during the index construction, and send it to CS. This step is essential for enabling secure data searches on CS. When CS receives t from the user, it uses t to retrieve $\check{S}_t$from H_T and then sends $\check{S}_t$ to the user.

After the user receives $\boldsymbol{\check{S}_t}$, he begins to decrypt it after recalculating the first key $sk_e = PRF_{sk}(2 \parallel w)$,

$S_t = Dec_{sk_e}(\check{S}_t)$. The next step is to set up a counter c and a list L_{id} to keep track of the end result. Subsequently, every letter in S_t is examined to ascertain if the word exists in the identifiers, and the identifiers are determined based on these cases.

5 Security analysis

In this part, we discussed the resistance of our scheme, the most famous types of attacks on SSE.

- **Frequency analysis attack**

A frequency analysis attack is a well-known ciphertext attack. It is based on examining the frequency of individual letters or groups of letters in a ciphertext. [6]. Therefore, it exploits the frequency of encrypted data uploaded to SI, which is either term frequency TF or term frequency-inverse TF_IDF. TF is defined as the number of times a word W appears in a document id, and $TF-IDF$ is the product of the term frequency (TF) and the inverse document frequency (IDF). IDF is calculated by dividing the total number of documents Ndb by the number of documents that contain the word n.

If the cloud server CS can access this critical information, it can execute this attack and identify the keyword being searched for.

Based on the above, we can conclude that our work is secure against this type of attack because the values stored in the cloud server CS are encrypted and do not directly reveal the original identifiers. Instead, they are transformed into an obscure text that enables the user to access the identifiers later.

- **IKK attack**

The IKK attack utilizes disclosed partial information to determine the plaintext words associated with the user search trapdoors. Consequently, this attack primarily depends on the leakage of access pattern information, which is defined as the outcome of the cloud server's CS search for t in SI, For instance, let's consider our database focuses on computer science, and a user submits three queries as trapdoors:t1, t2, and t3, representing the words "Hardware," "software," and "information," respectively. Once the communication between the user and CS is completed, the CS examines the obtained results, which are the sets of identifiers corresponding to the trapdoors. The CS can then calculate the probability of any two of these keywords appearing in the same document by observing the number of documents that are returned for those corresponding trapdoors. By continuing the search and leaking the access pattern to obtain more probabilities, the server can determine the keywords that correspond to the trapdoors [6]. However, after clarifying this attack, we can assert that our scheme resists the IKK attack because the search results by the CS are encrypted, and the access pattern does not reveal any significant information. Thus, the CS cannot access the identifiers corresponding to the trapdoors [27].

- **Keyword guessing attack (KGA)**

Keyword guessing attack is an attack on the encrypted index stored on the server, where the attacker attempts to guess the keyword being searched in order to use it later to find its identifiers [28]. This attack can be mitigated by various precautionary measures, such as encrypting the keywords themselves and keeping the encryption key secret and secure. Both of these measures are implemented in our scheme. Therefore, we can state that our work is resistant to KGA attacks.

- **Man-in-the-middle attack**

This type of attack happens when the communication channel between the user and the CS is not secure, allowing the attacker to impersonate one of the parties [29].Our work resists a Man-in-the-middle attack, due to the secret channel between the two parties (user and server), as the copy of the secret key exists only with the user and the server.

6 Experimental results

In this section, we evaluate our scheme using a real-world database of Wikipedia articles. We executed our experiments on a Windows 64-bit machine running an Intel Core i5 CPU clocked at 1.6 GHz and 8GB RAM. The database contains 2,050 identifiers $Ndb = 2{,}050$ and 525,430 words $W = 525{,}430$. Additionally, we chose a database that supports locality to enable us to observe the effect on retrieval time due to its significant number of identifiers. Python was our language of choice for the implementation of the code, owing to its numerous features and popularity in the scientific community.

Comparison with previous studies

In this section, we will conduct a comparative analysis of our work with four previous studies that share a similar objective: to improve performance through locality. These studies are [[8], [9], [10], [30]] and we conducted all of these works at the beginning before starting the comparison process.

To compare the studies, we focused on the search time required for retrieving three words that vary in the number of identifiers. The first word, $W1$, has the highest number of identifiers ($n = 2025$), the second word, $W2$, has an average number of identifiers ($n = 1015$), and the last word, W3, has a very small number of identifiers ($n = 4$). The outcomes of acquiring $W1$ and $W2$ clearly demonstrate the disparity between our work and the prior works in terms of search speed, as shown in Fig. 2. Meanwhile, the results of retrieving W3 demonstrate that while our approach enhances the search of words with a substantial number of identifiers, it does not adversely impact the search of words with fewer identifiers as shown in Fig. 3. To ensure a fair comparison with previous works that did not include a $Find\_ids$ phase, we have included the time required for this phase along with the research time.

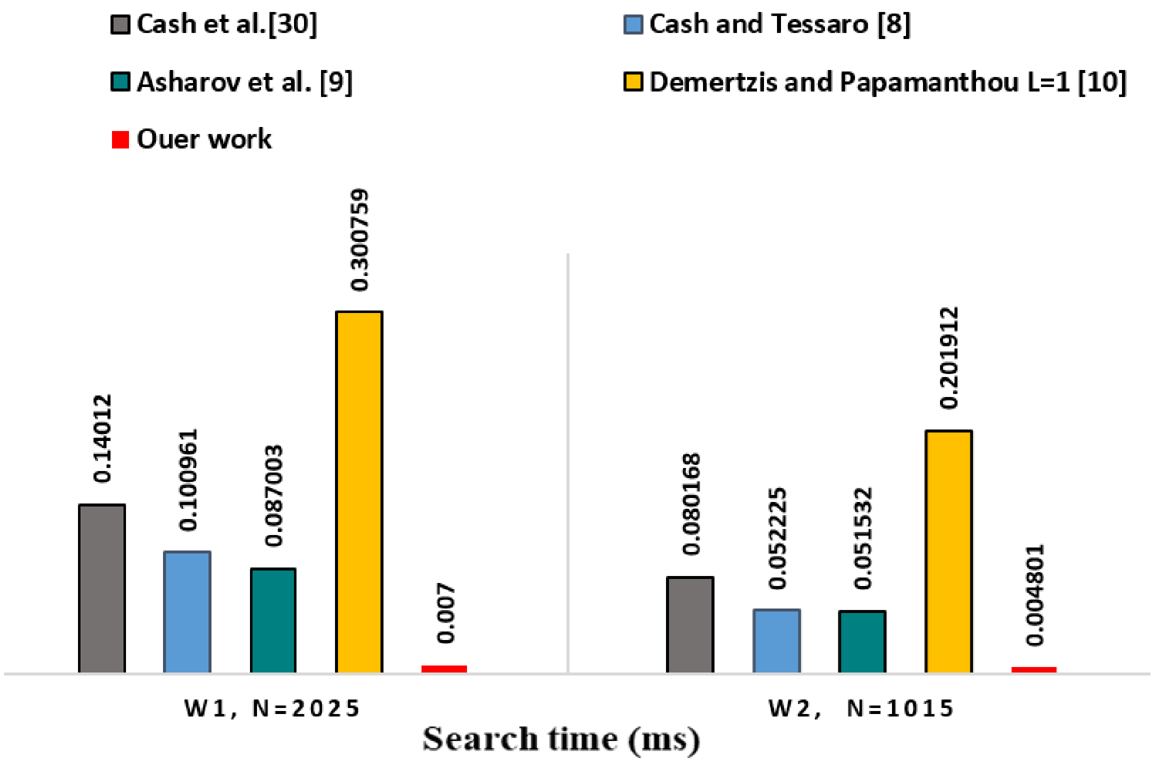

Figure 2: Comparing the search time required to find $W1$ and $W2$ in our scheme to that of previous schemes.

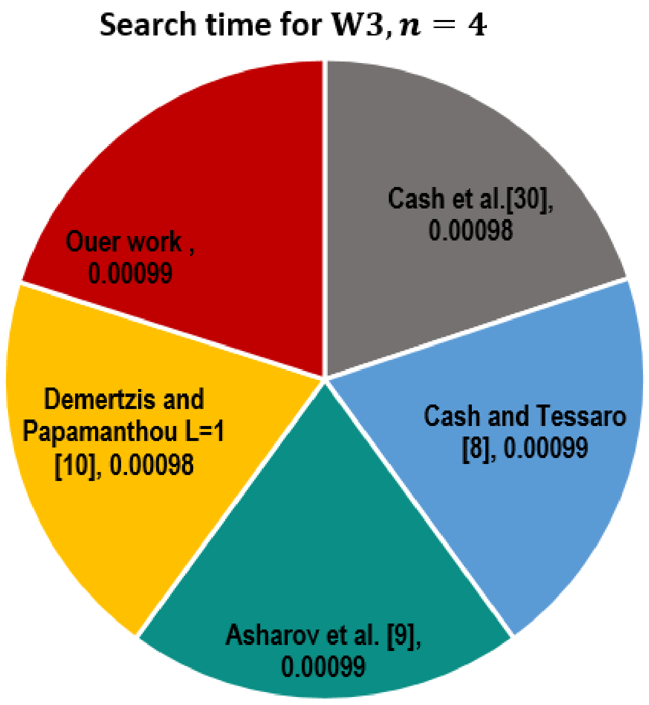

Figure 3: Comparison of search time for W3 in our scheme versus previous

6 Discussion

The primary reason for these promising results is the locality enhancements we integrated into our scheme, which significantly decreased the number of memory locations the cloud server needs to access to respond to a user's query. For example, in the work of Cash et al. [30], the cloud server had to traverse 2025 different memory locations to retrieve identifiers for the first word, whereas in our scheme, it only needs to access one location. Furthermore, the improved locality results in fewer decryption operations. In our scheme, decryption is performed once to obtain the identifiers, whereas in other schemes, multiple decryptions are required, affecting the overall speed of the results. Another factor contributing to these positive outcomes is that the cloud server in our schemes interacts with a single hash table, while in Asharov et al. (Scheme 3) [9], Cash and Tessaro [8], and Demertzis and Papamanthou [10], it manages multiple hash tables.

7 Conclusions

Our primary goal for this work is to enhance the overall performance of SSE. We are addressing a problem related to large DB, specifically the issue of poor locality caused by the many moves in memory by the cloud server during the search phase. We have made changes to *SI* storage mechanism, resulting in a significant improvement in the search performance. Our modification has optimized the locality to $O(1)$ without impacting the reading efficiency, which remains at $O(1)$. Additionally, this change has not resulted in significant increase in the storage space of the encrypted index.

Our work provides a high level of security since the server does not decrypt the data. Instead, it sends the encrypted data to the user, who decrypts it. Moreover, the values stored on the server are all equal in size and do not reveal the identifiers themselves. These values serve as evidence to obtain the identifiers later.

References

[1] M. Malathi, “Cloud computing concepts,” in *2011 3rd International Conference on Electronics Computer Technology*, 2011, vol. 6, pp. 236–239. doi: 10.1109/ICECTECH.2011.5942089.

[2] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, “Cloud computing,” *IBM white Pap.*, vol. 321, pp. 224–231, 2007.

[3] T. Dillon, C. Wu, and E. Chang, “Cloud computing: issues and challenges,” in *2010 24th IEEE international conference on advanced information networking and applications*, 2010, pp. 27–33. doi: 10.1109/AINA.2010.187.

[4] J. R. Vacca, *Cloud computing security: foundations and challenges*. CRC press, 2016.

[5] Y. Wang, J. Wang, and X. Chen, “Secure searchable encryption: a survey,” *J. Commun. Inf. networks*, vol. 1, pp. 52–65, 2016, doi: https://doi.org/10.1007/BF03391580.

[6] D. V. N. Siva Kumar and P. Santhi Thilagam, “Searchable encryption approaches: attacks and challenges,” *Knowl. Inf. Syst.*, vol. 61, no. 3, pp. 1179–1207, 2019, doi: https://doi.org/10.1007/s10115-018-1309-4.

[7] G. Sen Poh, J.-J. Chin, W.-C. Yau, K.-K. R. Choo, and M. S. Mohamad, “Searchable symmetric encryption: designs and challenges,” *ACM Comput. Surv.*, vol. 50, no. 3, pp. 1–37, 2017, doi: {10.1145/3064005}.

[8] D. Cash and S. Tessaro, “The locality of searchable symmetric encryption,” in *Annual international conference on the theory and applications of cryptographic techniques*, 2014, pp. 351–368. doi: https://doi.org/10.1007/978-3-642-55220-5_20.

[9] 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*, 2016, pp. 1101–1114. doi: {10.1145/2897518.2897562}.

[10] I. Demertzis and C. Papamanthou, “Fast searchable encryption with tunable locality,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1053–1067. doi: {10.1145/3035918.3064057}.

[11] G. Asharov, G. Segev, and I. Shahaf, “Tight tradeoffs in searchable symmetric encryption,” *J. Cryptol.*, vol. 34, no. 2, pp. 1–37, 2021, doi: https://doi.org/10.1007/s00145-020-09370-z.

[12] E.-J. Goh, “Secure indexes,” *Cryptol. ePrint Arch.*, 2003.

[13] Y.-C. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data,” in *International conference on applied cryptography and network security*, 2005, pp. 442–455.

[14] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: improved definitions and efficient constructions,” in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 79–88. doi: {10.1145/1180405.1180417}.

[15] R. Zhang, R. Xue, and L. Liu, “Searchable encryption for healthcare clouds: A survey,” *IEEE Trans. Serv. Comput.*, vol. 11, no. 6, pp. 978–996, 2017, doi: 10.1109/TSC.2017.2762296.

[16] 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*, 2000, pp. 44–55. doi: 10.1109/SECPRI.2000.848445.

[17] 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 *Annual cryptology conference*, 2013, pp. 353–373. doi: https://doi.org/10.1007/978-3-642-40041-4_20.

[18] M. Chase and S. Kamara, “Structured encryption and controlled disclosure,” in *International conference*

on the theory and application of cryptology and information security, 2010, pp. 577–594. doi: https://doi.org/10.1007/978-3-642-17373-8_33.

[19] P. Van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker, "Computationally efficient searchable symmetric encryption," in *Workshop on Secure Data Management*, 2010, pp. 87–100. doi: https://doi.org/10.1007/978-3-642-15546-8_7.

[20] K. Kurosawa and Y. Ohtaki, "How to update documents verifiably in searchable symmetric encryption," in *c*, 2013, pp. 309–328. doi: https://doi.org/10.1007/978-3-319-02937-5_17.

[21] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 965–976. doi: {10.1145/2382196.2382298}.

[22] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *International conference on financial cryptography and data security*, 2013, pp. 258–274. doi: https://ia.cr/2013/832.

[23] A. A. Alyousif and A. A. Yassin, "Locality Improvement Scheme Based on QR Code Technique within Inverted Index," *Informatica*, vol. 47, no. 7, 2023.

[24] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, 1996, pp. 1–15. doi: https://doi.org/10.1007/3-540-68697-5_1.

[25] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2020.

[26] Y. Watanabe *et al.*, "How to make a secure index for searchable symmetric encryption, revisited," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 105, no. 12, pp. 1559–1577, 2022, doi: 10.1587/transfun.2021EAP1163.

[27] 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*, 2015, pp. 668–679. doi: {10.1145/2810103.2813700}.

[28] 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 Trans. Cloud Comput.*, vol. 10, no. 2, pp. 835–848, 2020, doi: 10.1109/TCC.2020.2989296.

[29] S. Gangan, "A review of man-in-the-middle attacks," *arXiv Prepr. arXiv1504.02115*, 2015, doi: https://doi.org/10.48550/arXiv.1504.02115.

[30] D. Cash *et al.*, "Dynamic searchable encryption in very-large databases: Data structures and implementation," *Cryptol. ePrint Arch.*, 2014, doi: 10.14722/ndss.2014.23264.