Journal of Basrah Researches (Sciences) **49**(1), 102 (2023)
DOI: https://doi.org/10.56714/bjrs.49.1.9

Improving the Performance of Searchable Symmetric Encryption by Optimizing Locality

Aya A. Alyousif [1], Ali A. Yassin[1*], Zaid A. Abduljabbar[1,2], Ke Xu[2,3,4]

[1]Department of Computer Science, College of Education for Pure Sciences, University of Basrah, Basrah, Iraq.
[2]Shenzhen Research Institute of Huazhong University of Science and Technology, Shenzhen,China.
[3]College of Computer Science, South-Central Minzu University, Wuhan, China.
[4]Shenzhen Virtual University Park, Shenzhen, China.

ARTICLE INFO

Received 25 April 2023
Accepted 10 June 2023
Published 30 June 2023

Keywords:

Locality, Optimal locality, Searchable Symmetric Encryption, Cloud Server, Inverted Index,

Citation: Aya A. Alyousif et al., J. Basrah Res. (Sci.) **49** (1), 102 (2023).
DOI:https://doi.org/10.56714/bjrs.49.1.9

ABSTRACT

Both individuals and organizations prioritize data security and privacy, leading to an increasing focus on technological solutions that emphasize these aspects. Searchable symmetric encryption stands out as a prominent choice for secure storage and search of encrypted data on cloud servers, contributing to this objective. While searchable symmetric encryption offers numerous benefits, it also faces certain challenges, particularly when dealing with large databases. One significant challenge is poor performance, often attributed to poor locality. Visiting multiple locations can significantly increase the time required for data retrieval. Additionally, optimization methods aimed at improving locality can sometimes impact read efficiency or result in excessive storage of the encrypted index on the cloud server. In this paper, we present a scheme that effectively addresses these issues. We have enhanced the encrypted inverted index storage mechanism to improve information retrieval performance. Our scheme achieves optimal locality O(1) and optimal read efficiency in O(1), resulting in a significant increase in retrieval speed. Through experimentation with real-world data, we have demonstrated the practicality, accuracy, and security of our scheme.

1. Introduction

In the era of information technology and Internet, we are generating more data than ever before, and this trend is only set to continue. For businesses, individuals, and organizations, the ability to store, access, and manage this data efficiently is essential. In response, cloud storage has become a popular solution due to its many advantages over traditional forms of data storage [1][2]. It is a method of storing data on remote servers that can be accessed from any internet-connected device anywhere in the world. This results in data being stored in a centralized location that is easily

***Corresponding author email :** ali.yassin@uobasrah.edu.iq

ISSN: 1817-2695 (Print); 2411-524X (Online)
Online at: https://jou.jobrs.edu.iq

accessible. The greatest benefit of cloud storage is its flexibility of storage. Unlike traditional storage methods, it is not restricted by physical storage devices and offers virtually limitless storage capacity. This makes it an excellent choice for all parties involved in the storage process, from businesses to individuals.

Cloud storage offers not just superior storage capabilities, but also exceptional reliability. Cloud storage providers utilize redundant storage systems and multiple data centers located in different geographic locations, guaranteeing data accessibility even in the face of natural disasters or hardware failures. With cloud storage, companies can rest assured that their data is secure and accessible at all times, precisely when they need it.

To guarantee the security of data kept on cloud servers, a variety of techniques must be utilized, such as access management, network protection, and encryption [3]. Access management is a system that restricts data access based on user identity, position, or authorization. Network security includes safeguarding the network infrastructure used for data transfer. Encryption, conversely, is the procedure of converting data into a code to avoid unauthorized access. Encryption can be implemented both while in transit and at rest, ensuring that data remains safe throughout transmission and storage.

Several encryption techniques are available to secure data stored in the cloud server, including symmetric encryption, hashing, and searchable symmetric encryption (SSE) [4]. SSE is an encryption technique that enables authorized users to search for specific information in an encrypted database without compromising the confidentiality of the data to unauthorized individuals. However, SSE encounters various obstacles. It is a comparatively intricate procedure that entails searching within encrypted data and requires constructing and maintaining an index structure. This can consume significant time and resources. Moreover, there are security concerns related to the possibility of exposing confidential information [5].

SSE has recently been found to suffer from performance degradation and weakness in the retrieval process when handling huge databases. Further research has revealed that the cause of this issue is not related to the encryption process, but rather to the method used to store the index in memory. Storing the index structure in memory forces the cloud server to perform a series of memory transitions, commonly referred to as "poor locality" [4][6][7][8][9], during the search process for a user's query. These transitions can significantly slow down the retrieval of encrypted data and adversely affect the performance of SSE. Numerous researchers have shown interest in enhancing the performance in huge databases through locality optimization. However, this optimization process has been observed to potentially have negative impacts on both the storage space of data and the read efficiency of data. We can summarize our contributions as follows:

- Our scheme enhances information retrieval performance for all databases, regardless of their scale, by improving locality.
- Our scheme achieves optimal locality of $O(1)$, which implies that during each search operation, the cloud server only needs to access a single memory location, as opposed to multiple locations.
- Our suggested scheme is exceptionally secure as the server looks for the required data and delivers it to the user without decrypting it.
- Our scheme does not greatly increase the size of the encrypted index stored in the cloud server.
- Lastly, our scheme is highly read efficiency $O(1)$, where the cloud server only responds with the requested data when the user queries it.

Table 1. Symbols

Character	Description
w	Word
n	Number of words
W_{db}	Words in DB, $W_{db} = \{w_1, \ldots, w_n\}$
id	Identifier
IDs	Identifiers
ndb	Total IDs of DB
nw	Total of identifiers w
N	$\sum_{i=1}^{n} \lvert DB_w(w) \rvert$ where $DB_w(w) = \{id_1, .., id_{nw}\}$
HT	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"[9].
Add	Algorithm adds pairs of $(key, value)$ to HT
Get	value=Get(key)
St	String
$\check{S}t$	Encrypted string
EHT	A hash table to store encrypted values $\check{S}t$
$S1$	Counter counts the number of continuous identifiers in which w appears
$S0$	Counter counts the number of continuous identifiers in which w not appears
k_w	Derivative key to encryption and decryption of St
li	Label is used to store and retrieve $\check{S}t$ in HT, $Add(li, \check{S}t)$, $\check{S}t = Get(li)$
Enc	Function to encryption St
Dec	Function to decryption $\check{S}t$
ids	List to store the final result

Table 2. Searchable symmetric encryption algorithm

Algorithm	Description
Key generation: $k_e \leftarrow Gen\_k(1^\lambda)$	The key generation algorithm generates the secret key k_e based on the input security parameter1^λ.
Constructing secure index: $SI \leftarrow Enc\_DB(k_e, DB)$	The secure index SI is constructed by taking the secret key k_e and the database DB as inputs to this algorithm.
Token generator: $\tau \leftarrow Trpdr(k_e, w)$	In this algorithm, the user creates the token τ to search for data or a specific word w.
Search: $r \leftarrow Search(\tau, SI)$	The process in this algorithm involves the cloud server searching for the required data in the secure index and returning the result r to the user. If the result is encrypted, the user must using $Find\_ids$ algorithm to access it.
Find identifiers: $ids \leftarrow Find\_ids(k_e, r)$	The user uses this algorithm to obtain the final result, which consists of word identifiers ids after performing any necessary processing and deception the result r.

2. Paper Organization

The format of this paper is outlined below. In Section 3, we will elaborate on the concepts of searchable symmetric encryption. Previous related works will be discussed in Section 4. Our proposed scheme will be presented in Section 5, while Section 6 will delve into the details of our experimental results. To conclude, we offer our conclusions in the last section, Section 7.

3. Searchable Symmetric Encryption (SSE)

SSE is a technique that enables searching encrypted data while maintaining privacy [[10], [11], [12]]. It uses symmetric key cryptography, where the same key is used for both encryption and decryption. SSE involves three main components: the data owner DW, the cloud server CS, and the users The process of SSE comprises five essential steps. See Table 2 for for more details.

Firstly, DW selects a secret key k_e to be used for both encryption and decryption. Secondly, DW generates a searchable index SI from database DB and subsequently applies encryption to SI to ensure its security. The SI is then uploaded to CS. Thirdly, when a user wants to search for data on the cloud server, they generate an encrypted query called a token τ, using the same secret key. This token is then sent to CS. Fourthly, CS receives τ from the user and searches for the data in SI. In the fifth step, CS returns the search results r to the user either encrypted, for the user to decrypt later or decrypted and sent to the user.

4. Related Works

In 2000, a technology was introduced to facilitate searching of encrypted data without requiring prior decryption. The system, named "Searchable Symmetric Encryption"[13], empowers users to conduct searches for particular keywords within encrypted data while keeping the content secure. This marked the genesis of SSE. After the introduction of this new technology, researchers conducted extensive studies across various fields, such as performance optimization. Their findings reveal that

the decline in performance is not caused by the technology itself but by the server's access to memory positions during user request processing. The increase in the encrypted index size results in more positions being accessed, ultimately leading to slower response times [14]. This case is referred to as poor locality. There are generally two approaches that can be used to classify known constructions.

The first approach has constant read efficiency and linear space, but it suffers from poor locality as stated in [12] and [14]references. The process involves allocating an array of size N and uniformly mapping N elements from the database into the array. To retrieve a list of IDs that contain a given w, each id is stored in the array with a pointer to the next id in the list. Unfortunately, this approach requires the server to visit random positions in the array for every id associated with w, resulting in inefficiency.

The second approach is efficient for locality and read efficiency, but it requires a lot of additional storage[[15], [16], [17], [18]. This approach involves allocating a large enough array and uniformly mapping each DB_w list into the array based on the length of DB_w without any overlaps among different lists. Retrieving DB_w for a given w is efficient because the server only needs to access a single random position and read all consecutive id entries. This results in optimal read efficiency and locality. However, the positions of the lists in the array may expose information about the underlying database's structure. As a result, padding must be used to conceal information about the lengths of the lists, resulting in a polynomial increase in space overhead.

We should emphasize that there is often a problem either with the storage capacity, which is typically extensive, or with the locality. This can lead to a decline in the response time of the cloud server, and at times there is a bad effect on the read efficiency. Therefore, creating a construction that enjoys optimal locality with little storage space and at the same time does not affect its read efficiency is not easy, and it may not be possible. Cash and Tessaro [6] discussed this in 2014 and established a minimum tradeoff among these three cases. They also created a new construction with improved locality to $O(log\ N)$ and the storage space was $O(N\ log\ N)$.

Gilad Asharov et al. improved upon Cash and Tessaro's construction in 2016 [7] by achieving $O(1)$ locality while maintaining the storage space at O(N log N). In 2017, Demertzis and Papamanthou [8] created two constructions. The first construction had locality $O(1)$ and used $O(N\ s)$ storage space, where s is the number of levels utilized to store IDs. This construction had a negligible impact on read efficiency but still demanded a substantial amount of storage space. The second construction provided tunable locality, allowing the DW to choose a parameter during the creation of their index, while operating within the same storage space as the first approach. In 2021, Asharov and colleagues [9] made noteworthy advancements by creating two comprehensive frameworks - the pad-and-split framework and the statistical-independence framework - that strengthened the lower bound set 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 [19]–[24].

5. **Proposed Scheme**

This section will provide a comprehensive explanation of our scheme, where the following construction shows our entire scheme. Then we will explain our scheme with more details. See construction**.**

CONSTRUCTION: Let $= \{DB_w(w_1),\ldots,DB_w(w_n)\}$,
$W_{db} = \{w_1,\ldots,w_n\}$; For $w \in W_{db}$ let $DB_w(w) = \{id_1,..,id_{nw}\}$ and ndb is total of identifiers DB. See Table 1.

$k_e \leftarrow$ Gen_k(1^λ):

1. Input Security parameter 1^λ
2. Compute the secret key k_e with PRF
3. Output k_e

$EHT \leftarrow$ Enc_DB(k_e, DB):

1. Input k_e and DB
2. Initialize empty HT , EHT and $l = 0$
3. For every $w \epsilon W_{db}$
   - $St = ""$, $S1 = 0$, $S0 = 0$
   - Sort DB_w
   - If 1 in DB_w
     - Add "y" to St
   - Else
     - Add "n" to St
   - For $i = 1$ to ndb
     - If i in DB_w
       - $S1 = S1 + 1$
       - If $i + 1$ not in DB_w or $i + 1 = ndb + 1$
         - Add $S1$ to St
         - $S1 = 0$
     - Else
       - $S0 = S0 + 1$
       - If $i + 1$ in DB_w or $i + 1 = ndb + 1$
         - Add $S0$ to St
         - $S0 = 0$
   - End for
   - $i = i + 1$
   - If $length(St) > l$
   - $l = length(St)$
   - Add (w, St) to HT

   End for

4. For every $w \epsilon HT$
   - $St = Get\ (w)$
   - padding St to l with "0"
   - $k_w = PRF_{k_e}(2 \parallel w)$
   - $\check{S}t = Enc_{k_w}(St)$ by AES256
   - Compute $li = PRF_{k_e}(1 \parallel w)$
   - Add $(li, \check{S}t)$ to EHT

   End for

5. Output EHT

$\boldsymbol{\tau \leftarrow Trpdr(k_e, w)}$**:**
1. Input k_e and w
3. Compute $\tau = PRF_{k_e}(1 \parallel w) = li$
4. Output τ

$\boldsymbol{\check{S}t \leftarrow Search(\tau, EHT)}$**:**
1. Input τ and EHT
2. $\check{S}t = Get\ (li)$
3. Output $\check{S}t$

$\boldsymbol{ids \leftarrow Find\_ids(k_e, \check{S}t)}$**:**
1.Input k_e and $\check{S}t$
2. $k_w = PRF_{k_e}(2 \parallel w)$ and $S_t = Dec_{k_w}(\check{S}_t)$
3. Initialize empty ids list and $i = 1$
4. c=0
5. If the first character in the string matches "n"
For from $i = 2$ to length St
If $St =$ "0"
Out
If i mod 2 is equal 0
convert $St[i]$ to int
$c = c + St[i]$
Else
j=c
convert $St[i]$ to int
For from j to $j + St[i]$
Add j to ids
$j = j + 1$
End for

Else
For from $i = 2$ to length St
If $St =$ "0"
Out
If i mod 2 is not equal 0
convert $St[i]$ to int
$c = c + St[i]$
Else
j=c
convert $St[i]$ to int
For from j to $j + St[i]$
Add j to ids
$j = j + 1$
End for

6. Output $ids = identifiers$

5.1. Key generation phase $k_e \leftarrow Gen\_k(1^{\lambda})$:

To begin, the data owner generates a secret key using the Pseudo-Random Function [[25], [26]], it will serve for both encryption and decryption purposes.

5.2. Constructing secure index phase $EHT \leftarrow Enc\_DB(k_e, DB)$:

In this phase, the data owner must construct an encrypted index EHT for the database. This index is created using the words found in the database W_{db} and their corresponding identifiers. So, that each w is represented by an encrypted string $\check{S}t$ that indicates whether it appears or not in the identifiers of the database. The plaintext string St is determined by group of the number of consecutive identifiers where the word appears and the number of consecutive identifiers where it does not appear, across all identifiers in the database. There are two types of St that represent words. The first type begins with the letter "y," indicating that the word appears in the first identifier of the database $id_1 = 1$ and the first number in St represents the total number of consecutive identifiers where the word appears. The second type starts with the letter "n," indicating that the word does not appear in the first identifier of the database and the first number added to St represents the number of consecutive identifiers where the word does not appear. To demonstrate the aforementioned concept, consider the following example:
If $ndb = 15$ and $DB_w(w) = \{1,2,3,4,7,8,10,11,12,13,14,15\}$ In this case St will be $St =$ "y 4 2 2 1 6", The word is associated with the first four consecutive identifiers $(1,2,3,4)$ represented by the number 4, while the following two identifiers $(5, 6)$, represented by the number 2, are not associated with the word. The subsequent two identifiers (7, 8), represented by the number 2, are associated with the word, and this pattern continues sequentially until all the identifiers are covered. In conjunction generating St for all the words, the data owner must calculate the longest St. This step is crucial as it enables the padding of all St values to match the length of the longest value. This process enhances security and mitigates the risk of any data leakage to the cloud server. In sync with the padding procedure, the data owner will generate a key $k_w = PRF_{k_e}(2 \parallel w)$ and a label $li = PRF_{k_e}(1 \parallel w)$. The k_w is used to encrypt St value $\check{S}t = Enc_{k_w}(St)$ and li is used to store $\check{S}t$ in EHT.
The data owner can upload EHT to the cloud server once it has been constructed.

5.3. Token generator phase $\tau \leftarrow Trpdr(k_e, w)$ *and* Search phase $\check{S}t \leftarrow Search(\tau, EHT)$:

If a user wants to search for a specific word, they must create a token τ that represents the li generated during index construction and send it to the cloud server. Enabling secure data searching by the cloud server is dependent on this step. The cloud server can retrieve $\check{S}t$ from EHT once it receives τ from the user $\check{S}t = Get\ (li)$ and send $\check{S}t$ to user.

5.4. Find identifiers phase $ids \leftarrow Find\_ids(k_e, \check{S}t)$:

Upon receipt $\check{S}t$, the user initiates the decryption process by recomputing k_w and use it for decrypting $S_t = Dec_{k_w}(\check{S}_t)$. Once the S_t is obtained, the user will start processing it to find out which identifies the word belongs to, according to the steps shown in our construction.

6. Experimental Results

In this section, our scheme is evaluated using a real-world DB of Wikipedia articles. We conducted our experiments on a 64-bit Windows machine equipped with an Intel Core i5 CPU clocked at 1.6 GHz and 8GB RAM. Our database has $ndb = 2030$ and $w = 555{,}370$. We specifically selected DB that supports locality because it contains a significant number of IDs, enabling us to observe the impact on retrieval time. We chose to implement the code in Python due to its rich feature set and widespread use in the scientific community.

6.1. Comparison with previous schemes

In this section, we will compare our work to four previous schemes that share a similar goal of improving locality to enhance performance. These schemes are [[6], [7], [8], [27]]. We executed all of these schemes at the outset, prior to beginning the comparison process. In our comparisons, we focused on searching three words w, that vary in the quantity of their identifiers nw. The first word

w_1, had $nw = 2010$, the second word w_2 had $nw = 997$, and the last word w_3, had $nw = 3$. The results of searching for w_1 and w_2 provide clear evidence of the notable difference in searching speed between our work and previous schemes as shown in "Fig. 1". Moreover, the searching time for w_3 indicates that our approach enhances the search of words, with a significant number of identifiers without compromising the search of words with fewer identifiers as shown in "Fig. 2". 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.

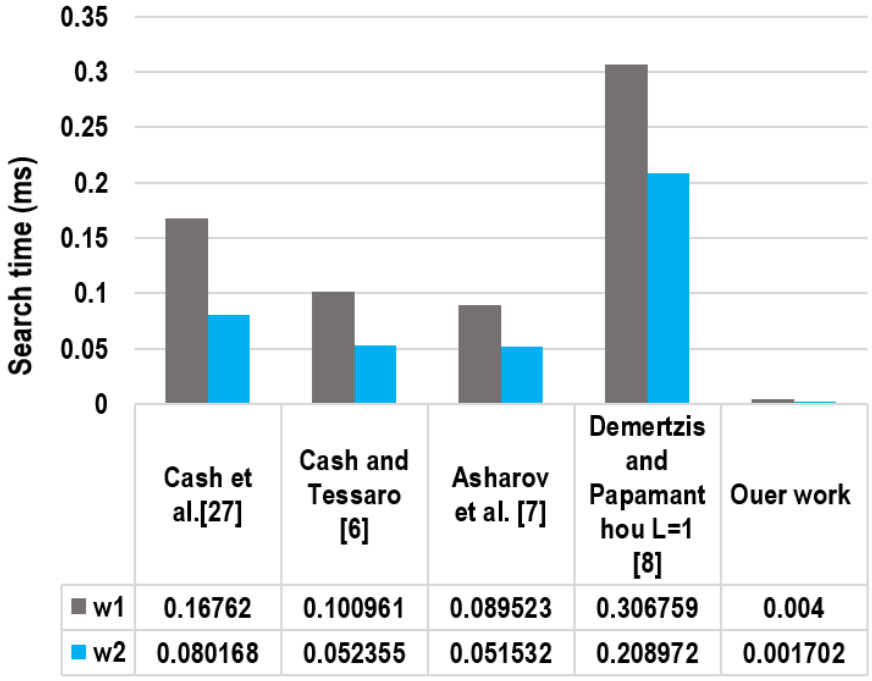

Fig. 1: Comparing the search time required to find $W1$ and $W2$ in our scheme to that of previous

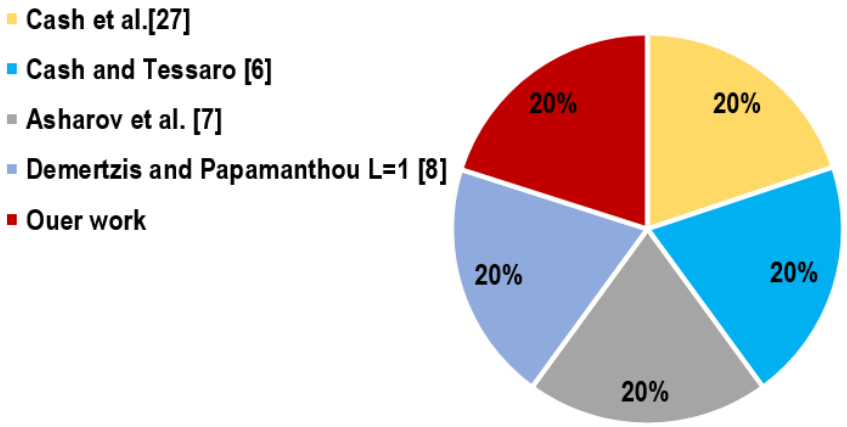

Figure 2: Comparison of search time for W3 in our scheme versus previous schemes.

7. Conclusion

Our main objective for this work is to improve the overall performance of the searchable symmetric encryption. Specifically, we are targeting a problem related to large databases, namely the issue of poor locality caused by frequent memory movements by the cloud server during the search phase. To solve this problem, we have implemented modifications to the inverted index storage mechanism, leading to a significant improvement in the search performance in the large database. Our modifications have optimized locality to $O(1)$ without any impact on the reading efficiency, which continues to be at $O(1)$. Furthermore, this change has not led to a significant increase in the encrypted index's storage space. Our work ensures a high level of security since the server does not decrypt the data but forwards the encrypted data to the user for decryption. Furthermore, the values stored on CS uniform in size and do not reveal IDs themselves. These values serve as evidence to retrieve IDs later.

8. References

[1] A. Ahmed, S. Kumar, A. A. Shah, A. Bhutto, Trop. Sci. J.,**2**(1), 1(2023).
[2] M. F. Mushtaq, U. Akram, I. Khan, S. N. Khan, A. Shahzad, A. Ullah, Int. J. Adv. Comput. Sci. Appl., **8**(10), 2017.
[3] J. R. Vacca, CRC press, 2016.
[4] G. Sen Poh, J.-J. Chin, W.-C. Yau, K.-K. R. Choo, M. S. Mohamad, ACM Comput. Surv., **50**(3), 1 (2017).
[5] D. V. N. Siva Kumar, P. Santhi Thilagam, Knowl. Inf. Syst., **61**(3), 1179(2019).
[6] D. Cash S. Tessaro, in Annual international conference on the theory and applications of cryptographic techniques, 351(2014).
[7] G. Asharov, M. Naor, G. Segev, I. Shahaf, in Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, 1101(2016).
[8] I. Demertzis ,C. Papamanthou, in Proceedings of the 2017 ACM International Conference on Management of Data, 1053(2017).
[9] G. Asharov, G. Segev, I. Shahaf, J. Cryptol., **34**(2),1(2021).
[10] E.-J. Goh, Cryptol. ePrint Arch., 2003.
[11] Y.-C. Chang, M. Mitzenmacher, in International conference on applied cryptography and network security, 442(2005).
[12] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, in Proceedings of the 13th ACM conference on Computer and communications security, 79(2006).
[13] D. X. Song, D. Wagner,A. Perrig, in Proceeding 2000 IEEE symposium on security and privacy. S&P 2000, 44(2000).
[14] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, M. Steiner, in Annual cryptology conference, 353(2013).
[15] M. Chase, S. Kamara, in International conference on the theory and application of cryptology and information security, 577(2010).
[16] P. Van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, W. Jonker, in Workshop on Secure Data Management, 87(2010).
[17] K. Kurosawa, Y. Ohtaki, in c, 309(2013).
[18] S. Kamara, C. Papamanthou, T. Roeder, in Proceedings of the 2012 ACM conference on Computer and communications security, 965(2012).
[19] H. M. Mohammed, A. I. Abdulsada, Iraqi J. Electr. Electron. Eng., **17**(2), 2021.
[20] H. M. Mohammed, A. I. Abdulsada, J. Basrah Res., **47**(1), 2021.
[21] C. Guo, W. Li, X. Tang, K.-K. R. Choo, Y. Liu, IEEE Trans. Dependable Secur. Comput., 2023.
[22] Z. A. Abduljabbar, A. Ibrahim, M. A. Al Sibahee, S. Lu, S. M. Umran, in 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), 1397(2021).
[23] M. A. Al Sibahee, A. I. Abdulsada, Z. A. Abduljabbar, J. Ma, V. O. Nyangaresi, S. M. Umran, Appl. Sci., **11**(24), 2021.
[24] M. A. Hussain et al., Egypt. Informatics J., **23**(4), 145(2022).

[25] M. Bellare, R. Canetti, H. Krawczyk, in Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings **16**, 1(1996).
[26] J. Katz,Y. Lindell, CRC press, 2020.
[27] D. Cash et al., Cryptol. ePrint Arch., 2014.

Journal of Basrah Researches (Sciences) **49**(1), 102 (2023)
DOI: https://doi.org/10.56714/bjrs.49.1.9

تحسين التشفير المتماثل القابل للبحث بواسطة المحلية المثلى

اية عبد الحسين اليوسف[1]، علي عادل ياسين*[1]، زيد امين عبد الجبار[3,1]، Ke Xu[2,3,4]

[1]قسم علوم الحاسوب، كلية التربية للعلوم الصرفة، جامعة البصرة، البصرة، العراق.
[2]كلية علوم الحاسوب ، جامعة الجنوب المركزية مينزو ، ووهان ، الصين.
[3]معهد شينزين للبحوث بجامعة هواتشونغ للعلوم والتكنولوجيا ، شينزين ، الصين.
[4] جامعة شينزين الافتراضية ، شينزين ، الصين.

معلومات البحث

الاستلام 25 نيسان 2023
القبول 10 حزيران 2023
النشر 30 حزيران 2023

الكلمات المفتاحية

المنطقة المحلية ، المنطقة المحلية المثلى ، التشفير المتماثل القابل للبحث ، الخادم السحابي، الفهرس المقلوب.

Citation: Aya A. Alyousif et al., J. Basrah Res. (Sci.) **49** (1), 102 (2023). DOI:https://doi.org/10.56714/bjrs.49.1.9

الملخص

يضع كل من الأفراد والمؤسسات قيمة عالية لحماية أمان وخصوصية بياناتهم. نتيجة لذلك، هناك تركيز متزايد على الحلول التكنولوجية التي تعطي الأولوية للخصوصية والأمان، يعد التشفير المتماثل القابل للبحث، أحد اهم الخيارات البارزة في هذا المجال. على الرغم من فوائده العديدة يواجه SSE بعض التحديات، خاصة عند التعامل مع قواعد البيانات الكبيرة. أحد التحديات مع SSE هو الأداء الضعيف، غالبًا بسبب المنطقة المحلية الضعيفة التي تكون نتيجة لزيارة الخادم السحابي لمواقع ذاكره متعددة عند البحث عن البيانات تؤدي لزيادة كبيرة في الوقت المطلوب لاسترداد البيانات. بالإضافة إلى ذلك، يمكن أن تؤثر طرق التحسين التي تهدف إلى تحسين الموقع في بعض الأحيان على كفاءة القراءة أو تؤدي إلى تخزين مفرط للفهرس المشفر المخزن على الخادم السحابي. في هذا البحث، نقدم مخططاً يعمل على حل هذه المشكلات بنجاح. علاوة على ذلك، قمنا بتحسين آلية تخزين الفهرس المقلوب المشفر لتحسين أداء استرجاع المعلومات. يحقق مخططنا الموقع الأمثل وكفاءة القراءة في O (1)، مما يؤدي إلى زيادة كبيرة في سرعة الاسترجاع. أثبتت تجربتنا مع بيانات العالم الحقيقي مدى التطبيق العملي والدقة والأمان لمنهجنا، مما يجعله حلاً موثوقًا به لاسترجاع المعلومات بشكل آمن وفعال.

***Corresponding author email :** ali.yassin@uobasrah.edu.iq

ISSN: 1817-2695 (Print); 2411-524X (Online)
Online at: https://jou.jobrs.edu.iq