Locality Improvement Scheme Based on QR Code Technique Within Inverted Index

Aya A. Alyousif*, Ali A. Yassin
E-mail: pgs.aya.alyousif@uobasrah.edu.iq, ali.yassin@uobasrah.edu.iq
Department of Computer Science, Education College for Pure Sciences, University of Basrah, Basrah, Iraq
*Corresponding author

Keywords: advanced encryption standard, locality, qr code, searchable encryption

Received: March 16, 2023

Searchable symmetric encryption is one of the most important modern technologies that allow the owner to store private data on an unreliable server and search for the data securely while preserving the data's confidentiality and privacy. This field has several schemes, but these schemes suffered from slower data retrieval in the case of large database sizes owing to the poor locality. Hence, the server visits several memory locations for a single query. Other studies focused on improving the locality, but the result is either increased storage capacity or decreased efficiency of data reading. In the present study, we present a secure, searchable scheme that overcomes the abovementioned issues and works to improve the locality by exploiting the QR code technique and the Advanced Encryption Standard algorithm. Furthermore, our work maintains read efficiency, helps reduce the risk of data breaches, and protects sensitive information from being accessed by unauthorized individuals. Moreover, the proposed scheme can resist cyber security attacks, such as frequency analysis attacks and keyword guessing attacks. Additionally, we used real-world data in our experiments and demonstrated that our proposed scheme is secure and practically efficient and holds high accuracy.

Povzetek: Predstavljana je varnostna iskalna shema za izboljšanje šifriranja, ki izboljšuje lokalnost, ohranja učinkovitost branja in je odporna na kibernetske napade.

1 Introduction

Cloud storage outsourcing is a service in which a company outsources the storage of its data to a cloud storage provider. Cloud storage outsourcing can provide several benefits, including cost savings, increased scalability and flexibility, and improved data security and availability. Cloud storage providers typically have strict security measures to protect customer data, including measures such as encryption, secure data centers, and access controls. However, the data owner (DW) or company is still responsible for ensuring that its sensitive data are protected and in compliance with relevant regulations or industry standards.

Searchable symmetric encryption (SSE) is one of the most important operations that can be applied on sensitive data in the outsourcing cloud [1]. Using searchable encryption to store sensitive data in the cloud can help to protect the data from unauthorized access while still allowing it to be searched and used in a controlled manner. However, carefully evaluating the security and performance tradeoffs of different searchable encryption approaches and ensuring that the chosen approach meets the DW's needs and security requirements are essential. In more detail, SSE is done by the DW as a first step to encrypt the data using a secret key, and then, it is sent to the server. After, the DW creates a secure index based on its own database. The encrypted data and a secure index are sent to the cloud storage server, which could be the same untrusted server or another server chosen by the DW as a third party. The DW generates a search token used to retrieve information based on a secure index file to perform a secure search.

SSE schemes in the literature review have several syntaxes, which are divided into two types. Each type relies on the interaction between the server and the DW in each request query about data. The first type is a single-round interaction where the server also decrypts the data and sends the result to the DW (therefore, the server learns the output). Additionally, the second type uses more than one round of interaction (where the server learns no information about the output) [2]. We use the second type in our scheme for the security of the exchanged data over the communication channel between server and client. Therefore, the server cannot decrypt the data and preserve the privacy of the client's data. SSE schemes face several challenges as follows:

- **Key management**

In the SSE scheme, the same secret key is used for encrypting and decrypting the data. This case implies that the key must be safely distributed to all parties who need to be able to look for the encrypted data. This can pose a

difficulty, particularly in large organizations with many users [3].

- **Security**

Although symmetric searchable encryption schemes are generally secure, they are vulnerable to key compromise. If the secret key is compromised, an attacker could decrypt the encrypted data and potentially perform unauthorized searches.

- **Key size**

To ensure the security of the encrypted data, the secret key used in a symmetric searchable encryption scheme must be long enough to withstand brute-force attacks. This case can increase the size of the key, which can be a problem in systems with limited storage or bandwidth.

- **Performance**

Symmetric searchable encryption schemes can be slower than other encryption schemes because they require the use of a single shared key for encryption and decryption. This case can be a problem when searching large datasets as the search may take a long time to complete. Another issue related to the slow searching in large datasets is poor locality. Thus, locality is defined as the number of times a server accesses multiple memory locations separately in response to a single request from the user [1], [2], [4]–[6].

The improvement in locality always leads to a negative impact on other characteristics, such as the efficiency of reading or a significant increase in storage capacity. Hence, finding a scheme that combines the best locality and the efficiency of reading and storage is difficult.

Our contribution

In this study, we propose a good locality secure searchable encryption based on the QR code technology and symmetric encryption method to solve the problem of searchable encryption in large datasets. Additionally, we use a new inverted index to improve locality that contributes to preventing leakage data and preserves the privacy data of DW. Provided with a comprehensive description of our scheme and a discussion of its resistance against several of the most famous attacks on SSE. We will briefly summarize our contributions as follows:

- Optimization locality by 800 times for small databases that contain less than a thousand identifiers and by 400 times for large databases. Furthermore, we applied our proposed scheme in real-world data, and the experimental results denote that our work achieves the best results in performance, read efficacy, and resisting malicious attacks compared with previous works.
- Our proposed scheme does not excessively affect storage and maintains storage without resulting in a large increase due to locality improvement. Furthermore, it maintains read efficiency at $O(1)$.

Table 1: List of symbols

Character	Description
V	Number of users
U_i	User, where $i \in V$
w_i	Word
m	Number of words
W	Words in DB, $W = \{w_i, \ldots, w_m\}$
id	Identifier
n	Total of identifiers DB
n_i	Total of identifiers w_i
N	$\sum_{i=1}^{m} \lvert DB(w_i)\rvert$ where $DB(w_i) = \{id_1, .., id_{n_i}\}$
PRF	Pseudo-random function
H_T	Hash table representing data structures used to store and retrieve data and consist of a pair of algorithms Add and Get [6]
Add	Algorithm adds pairs of $(key, value)$ to H_T
Get	value=Get(key)
S	String
$\hat{S}$	Encrypted string
la	Label is used to store and retrieve $\hat{S}$ in H_T, $Add(la, \hat{S})$, $\hat{S} = Get(la)$
Enc	Function to encryption S
Dec	Function to decryption $\hat{S}$
k_1	Derivative key to create la
k_2	Derivative key to encrypted and decrypted
b	Block of word identifiers
Q_img	QR code image
$Creat\_q$	Create QR code function
$Convert\_s\_q$	Convert string to Q_img function
$Convert\_q\_s$	Convert Q_img to string function
$\check{T}$	Encryption T
$Output_{list}$	A list used by the CS to store the values encoded in it to respond to the user query
$read$	Read Q_img function
k_u	Key used to secure T and $Output_{list}$ exchanged

	between major components CS and U_i
E_Output_{list}	Encryption $Output_{list}$

2 General background

2.1 SSE algorithms

- $k \leftarrow Gen(1^{\lambda})$: Is a key generation algorithm that is run by the DW token as a security parameter 1^{λ} (input) and a secret key k used for encrypting/decrypting database DB.
- $SI \leftarrow Enc(k, DB)$: Is used for building a secure index file (SI) based on k and DB.
- $T \leftarrow Trpdr(k, w_i)$: This algorithm is more concerned with preventing the disclosure of information stored on the server as an encrypted list of keywords (SI). As a result, the server responds to the users' request (T) in a safe manner.
- $d \leftarrow Search(SI, T)$: Is a deterministic algorithm run by the server to search for the data d through a trapdoor T in the secure index SI. If d is encrypted, then we will need a resolve algorithm.
- $R \leftarrow Resolve(k, d)$: This algorithm is performed by the DW to recover an identifiers for the keyword. It takes a secret key k and a data point d as inputs and outputs of the final result R.

2.2 Locality and read efficiency

We must get acquainted with the concept of read patterns to understand locality in more detail. The search function for any SSE scheme by the server $d \leftarrow Search(SI, T)$ can be analyzed into a series of intervals $[a_1, b_1]$... $[a_v, b_v]$, where the server starts from the first interval $[a_1, b_1]$ to the last interval $[a_v, b_v]$ depending on T in SI. Moreover, we can express these intervals as a read pattern function $RdPat(SI, T)$. When only one interval is obtained, the scheme has the best possible locality [4], [6].

Definition (Locality *L*) An SSE scheme Π has locality (L) with each security parameter (1^{λ}, DB, and $w_i \in$ N). We use $ReadPat(SI, T)$ consisting of L intervals with probability 1, when T and SI are computed as follows: $k \leftarrow Gen(1^{\lambda}), SI \leftarrow Enc(k, DB)$ and $T \leftarrow Trpdr(k, w_i)$. Furthermore, if $L = 1$, the SSE scheme (Π) has perfect locality. However, the perfect locality is insufficient because we can simply transform any scheme into a perfect locality by the poor read efficiency, which is defined as reading only the required data from the server for each query[4].

Definition (Read Efficiency *E*) An SSE scheme Π is E-read efficient with each main parameters 1^{λ}, DB, and $w_i \in$ N. We have $ReadPat(SI, T)$ that consists of intervals of total length at most $E \cdot |BinEnc(DB(w_i))|$ bits, where $BinEnc(DB(w_i))$ represents binary coding of $DB(w_i)$. The concatenation of all keywords' identifiers signified as bit strings [4]. A perfect locality can also be obtained by violating storage efficiency by creating a perfect locality scheme but with excessive storage space overhead (the encrypted DB should not be much larger than the original DB).

2.3 Related works

In 2000, Song et al. defined SSE and provided efficient constructions [7]. These mechanisms securely store data on an untrusted server but cannot retrieve client's data. Many subsequent studies focused on this field, achieving SSE's basic principle. However, practical experiments with large databases revealed poor performance and degradation with increasing size. The reason is that they often suffer from a bottleneck problem [8]. The literature found that the bottleneck in these schemes was not caused by encryption but by the lower-level memory access issues in more specifically poor locality. The known constructions can be broadly categorized into two.

The first approach is characterized by linear space and constant read efficiency but poor locality in [8], [9]. An array of size N is allocated, and N elements of the database are uniformly mapped into the array. To recover a list of documents that contain a given keyword, each document identifier is stored in the array together with a pointer to the next document identifier in the list. Additionally, the first approach requires the server to access random locations in the array with the number of identifiers that the word appears in. This case is inefficient because of poor performance resulting from moving to a large number of different locations.

The second approach has optimal read efficiency and locality but at the cost of substantial space overhead [10], [12]. The strategy behind this approach is to allocate a sufficiently large array and uniformly map the list of word identifiers into a contiguous interval in the array by the length of word identifiers, without any overlaps among different lists. To efficiently retrieve a list for a given keyword, the server needs to access only a single random location and read all consecutive identifier entries, thereby resulting in optimal read efficiency and locality. However, the locations of the lists in the array reveal information about the structure of the underlying database. Therefore, padding must be applied to conceal information about the lengths of the lists, leading to a polynomial space overhead. Hence, creating a scheme with the best locality, storage, and read efficiency is a challenge, as David Cash and Tessaro in 2014 [4] proved that it is impossible for this to happen. They also set a lower bound on the tradeoff among these three criteria. In addition to their improvement on the locality by creating a scheme with a logarithmic locality ($log\ N$), the storage space is not good enough $O(N\ log\ N)$.

In 2016, Gilad Asharov et al. [2], in their third scheme, updated the locality of David cash and Tessaro scheme to become $O(1)$ with the same storage space.

In 2017, Demertzis and Papamanthou [5] created two schemes, the first with optimal locality and space $O(N\ S_l)$. S_l is the number of levels used to store data, but there was

an effect on read efficiency by a small percentage, and the storage space is still large. As the second scheme worked in the same storage space as the first scheme, it achieves to a tunable locality, which is chosen as a parameter by DW during the setup phase.

In 2021, Asharov et al. [6] significantly strengthened the lower bound of Cash and Tessaro by creating two general frameworks, the first pad-and-split framework and the second statistical-independence framework.

From 2021 to 2023, various research studies in SSE across different domains have emerged, highlighting numerous benefits. However, all of them still lack good locality, as evidenced by references [14]–[19].

Table 2 displays previous works in terms of three critical features: locality, reading efficiency, and storage space. It is worth noting that none of the works satisfy all three characteristics simultaneously. Certain searches exhibit poor locality, such as [8], [9], [13], where a search word containing 2,000 identifiers $n_i = 2000$, would require the cloud server to traverse through 2,000 distinct positions to fulfill the user's request.

As a result, the searchable symmetric encryption's overall performance is hampered. In certain prior research, the locality has been favorable, as in [2], [4]and [10].

In cases where the locality is $O(1)$, the cloud server can fulfill the user's request by moving to just one location. If $N = 1200$ and the locality is $O(log\ N)$, it would be 10, implying that the cloud server would need to traverse 10 distinct positions to fulfill the user's request. Despite the favorable locality in some prior research [2], [4] and [10], they encountered issues with large storage space. For instance, if $N = 1200$ and the storage space is $O(N\ log\ N)$, the storage space will expand tenfold compared to its original size, such as in [2] [4].

Additionally, in other cases, the word that belongs to the most extensive set of identifiers can impact the storage space where, the storage space $O((\ \mathrm{Max}|DB(w_i)|)n)$. If this word corresponds to all the identifiers in the database, that is, $n = n_i$ the size of the encrypted index can become extremely large, as in [10].

3 Discussion

In this section, we will present our work's position in relation to locality, and storage space. By leveraging QR code technology, we have successfully improved the locality to $O(nq)$ by grouping a set of identifiers together in a single Q_img. This has led to a significant reduction in the number of cloud server readings required. This has the added benefit of reducing the number of times the user needs to perform decryption operations, resulting in a more efficient and streamlined experience.

Regarding storage space, since $= \sum_{i=1}^{m} |DB(w_i)|$, according to the previous example, the value of $N = 1700$, and this is its value in general. As for our work, N depends mainly $\sum nq$ for all words, and in previous example $\sum nq = 5$. Each one of these five Q_img will be stored with a size that is similar to the storage size required to store 400 identifiers, this implies that N's value will be around 2000, which is in proximity to the desired value.

Table 2: Comparison with previous schemes based on space, locality, and read efficiency

Related works	Space	Locality	Read efficiency
Curtmola et al. [13]	$O(N)$	$O(n_i)$	$O(1)$
Kamara et al. [9]	$O(N)$	$O(n_i)$	$O(1)$
David cash et al. [8]	$O(N)$	$O(n_i)$	$O(1)$
Chase and Kamara [10]	$O((\ \mathrm{Max}\|DB(w_i)\|)n)$	$O(1)$	$O(1)$
P. van Liesdonk et al.[11]	$O(mn)$	$O(n_i)$	$O(1)$
Kamara and Papamanthou [12]	$O(mn)$	$O(n_i\ log\ n)$	$O(n\ log\ n)$
David cash et al. [4]	$O(N\ log\ N)$	$O(log\ N)$	$O(1)$
Asharov et al. (Scheme 3) [2]	$O(N\ log\ N)$	$O(1)$	$O(1)$
Demertzis and Papamanthou [5]	$O\ (N\ s_i)$	$O(L_d)$ Where L_d is a tunable locality	$O(\frac{N^{\frac{1}{s}}}{L})$
Asharov et al. (Pad-and-split scheme) [6]	$O(N\ log\ N\ /\ log\ L)$	$O(L_d)$ Where L_d depends on the scheme in which it is implemented within its framework	$O(1)$
Our work	$N = \sum_{i=1}^{m}(\sum_{j=1}^{nq} S)$	$O(nq)$	$O(1)$

3.1 Primitive tools

- **QR codes**

A QR code is a two-dimensional code containing information in all directions, which is a digital image that can be easily scanned. Once scanned, it will quickly direct to the information embedded in the code, and the person who looks at the QR code is unable to identify it because its content is only machine-readable [20].

The QR code includes good storage capacity, fast readability, error correction, and support for more languages. It can hold 7,089 numeric characters and 4,296 alphanumeric characters [21].

- **Advanced Encryption Standard (*AES*)**

The AES algorithm is a widely used and recognized symmetric block cipher used for encrypting and decrypting sensitive data. It is implemented in hardware and software and is considered highly secure, making it difficult for hackers to access the original data. There is currently no known method for breaking AES encryption. AES offers the flexibility to use different key sizes, including 128, 192, and 256 bits, with a fixed block size of 128 bits.

AES is a block cipher algorithm that uses substitution and permutation network techniques to encrypt and decrypt data. It operates on a fixed plaintext block size of 128 bits (16 bytes) represented as a 4×4 matrix. The number of rounds in AES depends on the key size, with 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys [22].

4 Proposed scheme

In this section, we pay more attention to our proposed scheme, which consists of five phases as follows: key generation, setup, token generator, secure search, and resolve. There are three major components, DW, cloud server (CS), and user (U_i) to manage the environment of our work. Hence, $i \in V, V$ represents the number of users. The following construction explains the proposed scheme.

4.1 Our scheme construction with more detail

- **Key generation phase**

In this phase, DW creates a secret key k by PRF which it can be explained as follows:

Pseudo-Random Functions (PRF): A PRF function is $F: \{0,1\}^* \; x \; \{0,1\}^* \rightarrow \{0,1\}^*$. Thus, it receives two inputs a key and input. Therefore, F cannot be distinguished from a truly random function only with negligible probability in 1^λ, denoted as $negl(1^\lambda)$ [23]. It is to be used in setup and

CONSTRUCTION 1. (QR code based scheme)

Let $DB = \{DB(w_1), .., DB(w_m)\}$, $W =$ Words in $DB, W = \{w_i, \ldots, w_m\}$

For $w_i \in W$ let $DB(w_i) = \{id_1, .., id_{n_i}\}$.

Key generation phase k & $k_u \leftarrow Gen(1^\lambda)$:

1. Input Security parameter 1^λ
2. Output k and k_u, where k & $k_u \epsilon Z$
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. $bs = Fbs(n)$
4. Initialize empty H_T
5. For every $w_i \epsilon W$
   - $nq = n_i / bs$
   - Compute $k_1 = PRF_k(1 \parallel w_i)$ and $k_2 = PRF_k(2 \parallel w_i)$
   - Initialize counter $i = 0$
   - For every $b \epsilon nq$
     - $Q\_img = Create\_q(b)$
     - $S = Convert\_q\_s\ (Q\_img)$
     - Delete Q_img
     - $\hat{S} = Enc_{k_2}(S)$ by AES256
     - Compute $la = PRF_{k_1}(i)$
     - Add $(la, \hat{S})$ to H_T
     - $i = i + 1$
6. Upload H_T to CS

Token generator phase $\check{T} \leftarrow Trpdr(k_u, k, w_i)$:

1. Input k_u, k and w_i
2. Output $\check{T}$
3. Compute $T = PRF_k(1 \parallel w_i) = k_1$
4. $\check{T} = Enc_{k_u}(T)$
5. Send $\check{T}$ to CS

Secure Search phase $E\_Output_{list} \leftarrow Search(k_u, \check{T}, SI)$:

1. Input k_u, $\check{T}$ and SI
2. Output E_Output_{list}
3. Initialize counter $i = 0$ and $Output_{list}$
4. $T = Dec_{k_u}(\check{T}) = k_1$
5. While true
   - Compute $la = PRF_{k_1}(i)$
   - $\hat{S} = Get\ (la)$
   - Add$(\hat{S})$ to $Output_{list}$
   - $i = i + 1$

End While

6. $E\_Output_{list} = Enc_{k_u}(Output_{list})$
7. Send to $E\_Output_{list}\ U_i$

Resolve phase $R \leftarrow Resolve(k_u, k, E\_Output_{list})$:

1. 1.Input k , k_u and E_Output_{list}
2. 2.Output $R = identifiers$
3. $Output_{list} = Dec_{k_u}(E\_Output_{list})$
4. Restore $PRF_k(2 \parallel w_i) = k_2$
5. For every $\hat{S} \epsilon Output_{list}$
   $S = Dec_{k_2}(\hat{S})$
   $Q\_img = convert\_s\_q\ (S)$
   $R = read(Q\_img)$
   Delete Q_img
   END CONSTRUCTION

secure phase $SI \leftarrow Enc(k, DB)$ and send it to all trusted users.

U_i should use secret key k in the next phases, such as token generator phase $\check{T} \leftarrow Trpdr(k_u, k, w_i)$ and resolve phase $R \leftarrow Resolve(k_u, k, E\_Output_{list})$. In addition, a second key was created in the same way, called k_u, and it is used to secure T and $Output_{list}$ exchanged between major components CS and U_i.

▪ Setup and secure phase

The DW configures secure index SI, which will be uploaded to CS after completing this phase, where it uses k and DB as main inputs and computes SI as output. The following steps explain the mechanism working of the current phase:

- Configure empty hash table (H_T) that will be depended on to save secure parameters and then used as a secure index file (SI) for retrieving and processing the users' requests.
- Determine the size of the block bs = Fbs (n) where n represents the number of identifiers of the database (DB) and bs is the number of identifiers that can be stored together as a single block allowed to create Q_img. From experimental results of our work, we discover a significant relationship between the number of identifiers of database n and the size of Q_img. We found that the Q_img has the flexibility to save maximum numbers of identifiers, whereas the value of n is low based on the number of n's ranking)
- For every keyword taken from W where $w_i \epsilon W$, we carried out several successive steps as follows:
  - Compute the numbers of Q_img nq that w_i will be needed by n_i/bs, it is necessary to know the number of Q_img for each word based on the number of identifiers n_i of words after dividing it by bs.
  - Derivation of two keys from w_i. The first one k_1 is computed using PRF, which takes 1 and w_i as its inputs as follows: $k_1 = PRF_k(1 \parallel w_i)$. The label la has been created by k_1. The second key k_2 creates in the same way $k_2 = PRF_k(2 \parallel w_i)$.
  - Initialize counter $i = 0$.
  - **For every block** $b \epsilon nq$ do the following:
    - Convert this b to QR code image Q_img using the $Create\_q(b)$ function, which receives b as input and converts b to Q_img.
    - Convert the Q_img to a string S by using $S = Convert\_q\_s\ (Q\_img)$ function.
    - Delete Q_img
    - Encrypt S using $\hat{S} = Enc_{k_2}(S)$, $\hat{S}$ represents an encrypted string.
    - Derives la from w_i by $la = PRF_{k_1}(i)$, which receives k_1 and i as input to get a different la each time.
    - Add the la and $\hat{S}$ pair $(la, \hat{S})$ to the H_T as a (key, value).
    - Increase the value of i.

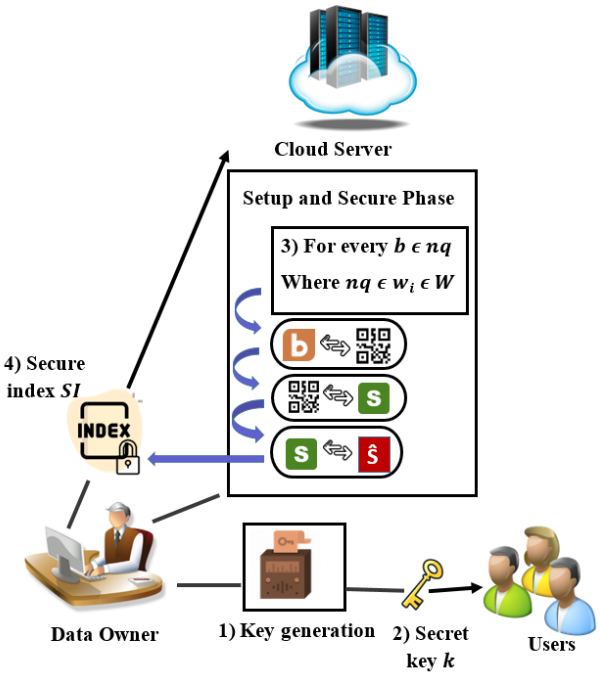


Figure 1: Key generation phase and setup and secure phase.

▪ Trapdoor phase

The current phase is implemented by U_i to search for a specific word w_i, the $Trpdr$ receives w_i , k_u and k as input parameters and gives $\check{T}$ as an output, which will compute T and then secure it using k_u, $\check{T} = Enc_{k_u}(T)$ to send it to CS more securely.

- **Secure search phase**

CS responds to a user's query and returns an encrypted list of secure values (E_Output_{list}) in SI. CS performs the following steps:

- $T = Dec_{k_u}(\check{T}) = k_1$
- Initialize counter $i = 0$ and empty $Output_{list} = []$.

While True
  Let $la = PRF_{k_1}(i)$
  Retrieve $Output_{list}[i] = \hat{S} = Get(la)$
  $i = i + 1$
End

- $E\_Output_{list} = Enc_{k_u}(Output_{list})$
- *CS* resubmits E_Output_{list} to U_i.

- **Resolve phase**

U_i receives E_Output_{list} from *CS* and performs the following:

- $Output_{list} = Dec_{k_u}(E\_Output_{list})$
- $k_2 = PRF_k(2 \parallel w_i)$
- **For every** $\hat{S} \in Output_{list}$
  - $S = Dec_{k_2}(\hat{S})$
  - Convert S to Q_img using the $convert\_s\_q\ (S)$ function.
  - Read the Q_img with the read function and get the identifiers stored in Q_img.

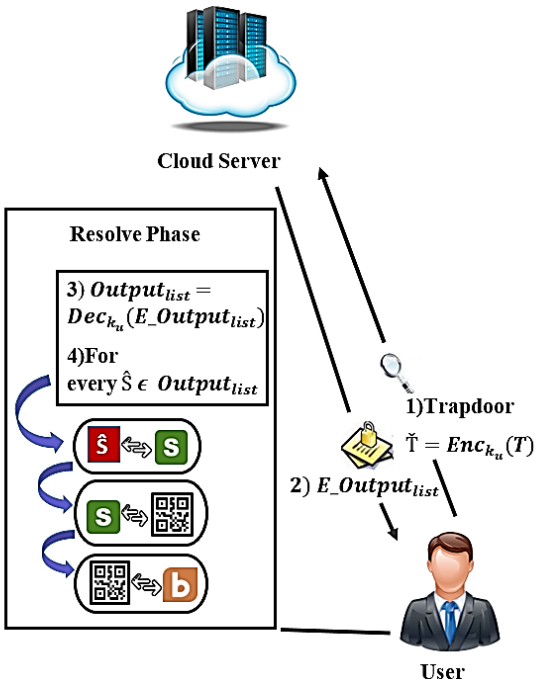


Fig. 2: Trapdoor, secure search and resolve phases.

5 Security analysis

5.1 Server information leakage

Leakage functions Ł, the amount of information the server can learn about the stored data. It can be divided into three types, the first two are $Ł_{max}$ and $Ł_{min}$ associated by single-round interaction SSE, and third one is $Ł_{size}$, which is in more than one round of SSE interaction. Then, all types take DB and W as input parameters [4], [6].

$Ł_{max}\ (DB, W)$: output $(N, \{DB(w_i)\}_{w_i \in W}, m, n, n_i, \text{Max}(DB(w_i))_{w_i \in W})$. It reflects the maximum cases of leakage.

$Ł_{min}\ (DB, W)$: output $(N, \{DB(w_i)\}_{w_i \in W})$. It is considered a somewhat acceptable leak. The server must know N in all cases. In a single-round interaction, the server will be able to view the query results because it can decrypt the result.

$Ł_{size}\ (DB, W)$: output $(N, \{|DB(w_i)|\}_{w_i \in W})$ in more than one round of interaction where the server cannot decrypt data. Thus, when the response to the user's request is encrypted, the server can learn only the output size, which is one of the best cases of leakage. In addition, detecting the size does not mean that the server has the ability to reveal the main information about our proposed scheme. Therefore, our work is resisting all types of leakage.

In our work, we go into deep details for security analysis. We noticed from practical experiments of our scheme that the shape of the Q_img is simple and less complicated if the number of word identifiers in the input b of the function is small $Q\_img = Create\_q(b)$. This case in turn will affect the next step, that is, the process of converting this Q_img into a string $S = Convert\_q\_s\ (Q\_img)$, which leads to a simple and less complex S. Its size differs from the size of another S coming from a block full of identifiers. This difference in size is considered a security vulnerability. However, we made sure to address it by making all strings have very close sizes. By increasing the complexity of the Q_img if the number of block identifiers is small, this Q_img will lead to a larger string size. Thus, the sum of S does not reflect the real size of the database because they are all close in size, even if they are the result of preparing different identifiers. In addition, our work is more than one round of interaction. Furthermore, the server responds to the user's request without decrypting the data. From this standpoint, we can say that our construction has only $Ł_{size}$ leakage.

5.2 Resisting attacks

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 one of the known ciphertext attacks. It is built on studying the frequency of letters or collections of letters in a ciphertext [24]. Hence, it takes advantage of 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 w_i looks in a document id, and TF-IDF is the multiplication of term frequency TF and IDF values. IDF is defined as a value of w_i that could be obtained by

dividing n by the document frequency number of documents in which such a word appeared. If the CS can access this important information, then it can carry out this attack and know the keyword that is being searched for.

Based on the above, we can say that our work is safe against the current attack because the values stored in CS are encrypted and do not reflect the identifiers originally, but rather an obscure text that helps the U_i to access identifiers later.

- ***IKK* attack**

IKK attack uses the disclosed partly information to find out what plaintext words the U_i searched trapdoors [25]. Hence, this attack relies mainly on the leaking of the access pattern information, which is defined as the result of the search for T in SI by CS.

For example, suppose our DB is about computer science, and the user requests three queries as trapdoors: T_1, T_2, and T_3, which represent the words "network," "computer," and "information," respectively. After completing the communication between U_i and CS, the CS will look at the obtained results, which is the set of identifiers corresponding to the trapdoors. The CS can then compute the probability of appearance of any two of these keywords in any document by observing the number of the same documents reverted for those corresponding trapdoors. By continuing the search and leaking the access pattern, thereby obtaining more probabilities, the server will be able to access the keywords that correspond to the trapdoors [24]. After a modest clarification of this attack, we can say that our scheme resists the IKK attack because the search result by the CS is encrypted, and the access pattern does not leak any important information. That is, the CS will not be able to access the identifiers $\{id_1, .., id_{n_i}\}$ corresponding to the trapdoors [25].

- **Keyword guessing attack (KGA)**

KGA attack is an attack on the encrypted index stored in the server, where the attacker tries to guess the keyword that is being searched to use it in the search later and find its identifiers [26]. This attack is resisted by various precautionary measures, such as encrypting the words themselves and keeping the key secret and secure. Both have been worked out in our scheme. Hence, we can say that our work resists KGA attacks.

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

This type of attack occurs when the communication channel between the U_i and the CS is not secured, as the attacker will occupy the identity of one of the parties (U_i, CS) [27]. We took the resistance to this attack into account in our work, where the communication channel was secured by encrypting the exchanged data (T and $Output_{list}$) between the CS and U_i based on k_u key, in addition to mutual authentication between the two parties with the same key as the following construction:

CONSTRUCTION 2.
In Trapdoor Phase:
Choose a random identifier u_{ID} by U_i
$hu_{ID} = H_{K_u}(u_{ID})$ by MD5
Send $(hu_{ID} \,||\, u_{ID} ||$ Ť) to CS
In Secure Search Phase:
Choose a random identifier cs_{ID} by CS
If hu_{ID}=$vrfy_{K_u}(u_{ID}) \rightarrow U_i$ authentication
Before send E_Output_{list}
$hcs_{ID} = H_{K_u}(cs_{ID})$ by MD5
send $(hcs_{ID} \,||\, cs_{ID} \,||\, E\_Output_{list})$ to U_i
In Resolve Phase:
If hcs_{ID}=$vrfy_{K_u}(cs_{ID}) \rightarrow CS$ authentication
Where H is hash function, $vrfy$ is verification function, hu_{ID} represent user hash value and hcs_{ID} represent cloud server hash value

6 Experimental results

In this section, we evaluated the performance of our proposed scheme using a real-data collection of Wikipedia articles. We selected a total of 250 articles, along with their eight corresponding historical versions, resulting in a collection of 2000 files. Our collection contained a total of 117000 keywords. As the total number of identifiers n for DB is 2000, the absorption of the QR code will be approximately 400 identifiers. In addition, the probabilities of nq values for each word will be from 1 to 5, which means that the maximum locality will be only 5. The experiments were conducted on a PC with a 2.6 GH Intel Core i5 CPU and 8 GB of RAM running on 64-bit Windows 10. The code was implemented in python because of its many known features, and it supports the creation of QR codes technologies.

- **Retrieval time**

We conducted an experiment to find out the time taken to retrieve identifiers for three words that differ in the number of their identifiers n_i and the number of QR code nq. Moreover, the first word w_1 contains 340 identifiers $nq = 1$, and the second word w_2 contains 1172 identifiers 1172/400 = 2.93. That is, 3 nq, and the third w_3 appears in all files 2000/400 = 5 nq. Evidently, the retrieval time grows linearly with the increase in nq values. We notice that the retrieval time is the time of each of the secure search and resolve phases.

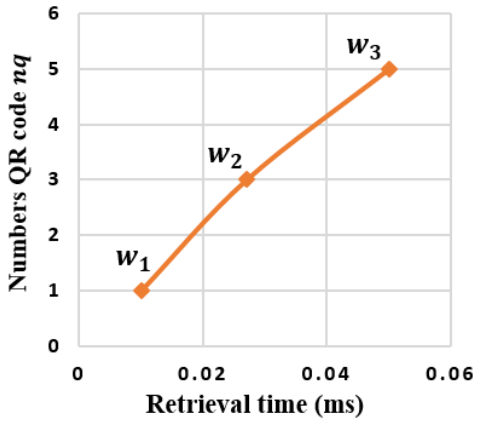
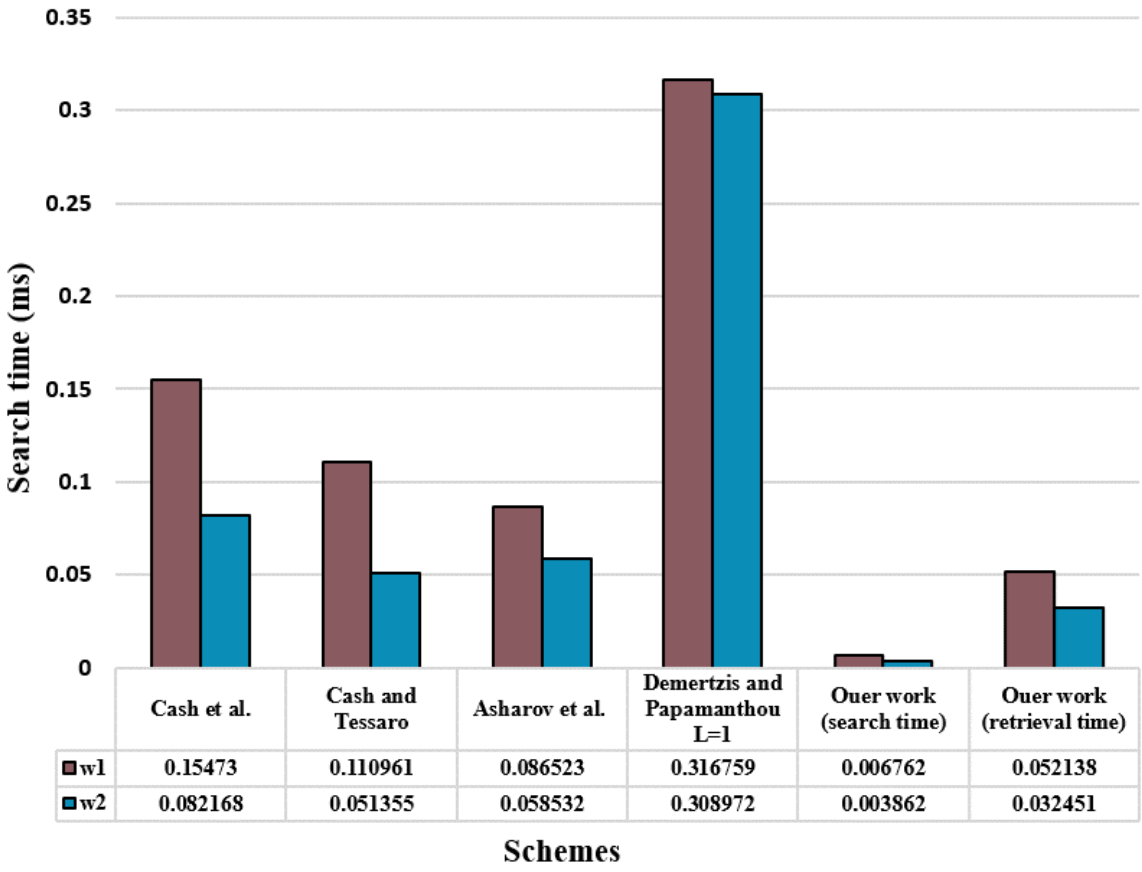


Figure 3: The retrieval time for three words.

- **Comparison with previous schemes**

To ensure the efficiency of our scheme, we implemented some of the previous schemes as a first step. These schemes are Cash et al. [28], Cash and Tessaro [4], Asharov et al. (Scheme 3) [2], and Demertzis and Papamanthou [5]. As a second step, we perform the same experiment on our scheme and these four schemes on the same database based on the AES256 encryption algorithm, which are the same words to know the time required to secure the search of the word identifiers. We chose two words: the first word w_1 with the largest number of identifiers and that appears in all database identifiers, that is, appears in 2000 files, and the second w_2 appears in 1000 files. For the comparison to be fair and as we used more than one round of interaction in our work to increase security, the search time was very short because CS does not decrypt and process the data. Hence, we calculated the retrieval time for our work to find out the total processing time to obtain the identifiers.



	Cash et al.	Cash and Tessaro	Asharov et al.	Demertzis and Papamanthou L=1	Ouer work (search time)	Ouer work (retrieval time)
w1	0.15473	0.110961	0.086523	0.316759	0.006762	0.052138
w2	0.082168	0.051355	0.058532	0.308972	0.003862	0.032451

Figure 4: Comparison with previous schemes.

7 Conclusions

In this study, we discuss the problem of slow retrieval of encrypted data owing to poor locality and present a new scheme that helps solve this problem by adding the QR code technique to the inverted index. This case mainly improves the locality. Finally, our scheme has less leakage and high security and is resistant to most common SSE attacks. Additionally, we used real-world data in our experiments and demonstrated that our proposed scheme is secure and practically efficient and holds high accuracy.

References

[1] 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}.

[2] 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}.

[3] E. Damiani, S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Key management for multi-user encrypted databases," in *Proceedings of the 2005 ACM workshop on Storage security and survivability*, 2005, pp. 74–83. doi: {10.1145/1103780.1103792}.

[4] 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.

[5] 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}.

[6] 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.

[7] 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.

[8] 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.

[9] 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}.

[10] 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.

[11] 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.

[12] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *International conference on financial cryptography and data security*, 2013, pp. 258–274.

[13] 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}.

[14] H. M. Mohammed and A. I. Abdulsada, "Secure Multi-keyword Similarity Search Over Encrypted Data With Security Improvement.," *Iraqi J. Electr. Electron. Eng.*, vol. 17, no. 2, 2021.

[15] H. M. Mohammed and A. I. Abdulsada, "Multi-keyword search over encrypted data with security proof," *J. Basrah Res.*, vol. 47, no. 1, 2021.

[16] 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 Trans. Dependable Secur. Comput.*, 2023, doi: 10.1109/TDSC.2023.3262060.

[17] Z. A. Abduljabbar, A. Ibrahim, M. A. Al Sibahee, S. Lu, and S. M. Umran, "Lightweight Privacy-Preserving Similar Documents Retrieval over Encrypted Data," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2021, pp. 1397–1398. doi: 10.1109/COMPSAC51774.2021.00202.

[18] M. A. Al Sibahee, A. I. Abdulsada, Z. A. Abduljabbar, J. Ma, V. O. Nyangaresi, and S. M.

Umran, "Lightweight, Secure, Similar-Document Retrieval over Encrypted Data," *Appl. Sci.*, vol. 11, no. 24, p. 12040, 2021, doi: https://doi.org/10.3390/app112412040.

[19] M. A. Hussain *et al.*, "Provably throttling SQLI using an enciphering query and secure matching," *Egypt. Informatics J.*, vol. 23, no. 4, pp. 145–162, 2022, doi: https://doi.org/10.1016/j.eij.2022.10.001.

[20] S. Singh, "QR code analysis," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 6, no. 5, 2016.

[21] A. S. Narayanan, "QR codes and security solutions," *Int. J. Comput. Sci. Telecommun.*, vol. 3, no. 7, pp. 69–72, 2012.

[22] A. M. Abdullah, "Advanced encryption standard (AES) algorithm to encrypt and decrypt data," *Cryptogr. Netw. Secur.*, vol. 16, pp. 1–11, 2017.

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

[24] 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.

[25] 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}.

[26] 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.

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

[28] 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.