OSM-DSSE: A Searchable Encryption Scheme with Hidden Search Patterns and Access Patterns

Hong Liu (liliary@163.com)
Chongqing University of Posts and Telecommunications https://orcid.org/0000-0002-1260-5360
Xueqin Li
Chongqing University of Posts and Telecommunications
Erchuan Guo
Chongqing University of Posts and Telecommunications
Yunpeng Xiao
Chongqing University of Posts and Telecommunications
Tun Li
Chongqing University of Posts and Telecommunications

Research Article

Keywords: Dynamic Searchable Encryption , Search Pattern , Access Pattern , Paillier Encryption , Differential Privacy

Posted Date: March 2nd, 2021

DOI: https://doi.org/10.21203/rs.3.rs-253711/v1

Noname manuscript No.
(will be inserted by the editor)

OSM-DSSE: A Searchable Encryption Scheme with Hidden Search Patterns and Access Patterns

Hong. Liu · Xueqin. Li · Erchuan. Guo · Yunpeng. Xiao · Tun. Li

Received: date / Accepted: date

Abstract Dynamic searchable encryption methods allow a client to perform searches and updates over encrypted data stored in the cloud. However, existing researches show that the general dynamic searchable symmetric encryption (DSSE) scheme is vulnerable to statistical attacks due to the leakage of search patterns and access patterns, which is detrimental to protecting the users' privacy. Although the traditional Oblivious Random Access Machine (ORAM) can hide the access pattern, it also incurs significant communication overhead and cannot hide the search pattern. These limitations make it difficult to deploy the ORAM method in real cloud environments. To overcome this limitation, a DSSE scheme called obliviously shuffled incidence matrix DSSE (OSM-DSSE) is proposed in this paper to access the encrypted data obliviously. The OSM-DSSE scheme realizes efficient search and update operations based on an incidence matrix. In particular, a shuffling algorithm using Paillier encryption is combined with 1-out-of-n obliviously transfer (OT) protocol and local differential privacy to obfuscate the search targets. Besides, a formalized security analysis and performance analysis on the proposed scheme is provided, which indicates that the OSM-DSSE scheme achieves high security, efficient searches, and low storage overhead. Also, this scheme not only completely hides the search and access patterns but also provides adaptive security against malicious attacks by adversaries. Furthermore, experimental results show that the OSM-DSSE scheme obtains 3-4x better execution efficiency than the state-of-art solutions.

Hong. Liu
School of Software Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
email: liliary@163.com

Keywords Dynamic Searchable Encryption · Search Pattern · Access Pattern · Paillier Encryption · Differential Privacy

1 Introduction

The rise of cloud service provides vast benefits to society and the IT industry. Storage-as-a-Service is one of the most common cloud services available, which allows the client to store data online remotely and access data everywhere, reducing the cost of data management and maintenance. Despite the merits, Storage-as-a-Service also brings significant security and privacy issues. Once data is outsourced, a client loses the ability to control the data. Also, sensitive information may be tampered or stole by a malicious user. Although the client can encrypt data with standard encryption schemes (e.g., AES) to ensure confidentiality, basic operations (e.g., search/update) on the encrypted data could not be performed. And substantial computational overhead is incurred, which greatly reduces the benefits of the cloud service.

To solve the above problems, in 2000, Song et al. [1] first proposed the concept of searchable symmetric encryption (SSE). As a new encryption primitive, sea-rchable encryption enables the user to search for a keyword over the ciphertext. However, the application was limited to search on static encrypted data and was unable to resist the simple adversary attack. In 2003, Goh et al. [2] formally defined the secure index and developed a security model called the "semantic security" for adaptive selective keyword attacks. However, the accuracy of query result was limited due to the use of the Bloom filter. In 2006, Curtmola et al. [3] proposed two new security models called "adaptive secu-

rity" and "non-adaptive security", introducing a single-keyword-search SSE with a formal security definition. Due to the limitations of the SSE proposed earlier and the dilemma between ensuring user privacy and efficient data usage on the cloud, Kamara et al. [4] introduced the dynamic searchable symmetric encryption (DSSE) method, which enabled the user to perform search and update operations on encrypted data.

The general searchable encryption algorithm improves the search efficiency at the cost of leaking some information about files or queries to the server, such as the search pattern and the access pattern. It is generally acknowledged that the searchable encryption scheme is secure unless it does not reveal user data and query information other than the information disclosed by the leakage profile. However, in the real world, an adversary can exploit these leakages to launch statistical attacks to recover the user data and query information. For instance, Islam et al. [5] and Cash et al. [6] firstly exploited access pattern leakage and prior knowledge about the dataset to recover the user's query information. Liu et al. [7] exploited the search pattern to launch attacks and obtained users' query information. Zhang et al. [8] completely exposed the client's query and recovered user data and query information through the file injection attack. Simon et al. [9] leveraged both access and search pattern leakages to recover the keywords of queries. Therefore, an important direction for future research is to focus on the suppression of information disclosure, rather than setting it as default.

For the leakages and attacks described above, although some solutions have been proposed, most of the research focuses on forward-secure and backward-secure methods [10–12]. The Oblivious Random Access Machine (ORAM) can address the problem of access pattern leakage [13–15], but it was impractical for wide-spread adoption. Garg et al. [16] exploited ORAM and garbled RAM (Random Access Memory) to hide the search pattern. Kamara et al. [17] proposed a general scheme for suppressing search pattern leakage. The united structured encryption (SE) based on ORAM made the scheme more efficient than ORAM, but the scheme was static. Besides, the recently proposed literature introduces differential privacy mechanisms [18] and hashing techniques [19] to obfuscate access patterns. In principle, solutions that do not leak any information to the server can be built on powerful techniques such as secure two-party computing, full homomorphic encryption (FHE), etc., but they are often impractical.

To achieve more secure searchable encryption, both the search pattern and the access pattern need to be hidden. The above-proposed solutions solve either the search pattern leakage or the access pattern leakage but not both. Although the method proposed by Hoang et al. [20,21] exploited distributed data structures to hide the search pattern or the access pattern, it is usually necessary to consider whether there is collusion between servers. Akavia et al. [22] proposed a secure search on an encrypted data structure using FHE. This scheme can seal the leakages of the search pattern and the access pattern, but it is difficult to deploy in real environments. In addition, these solutions only address the information leakage on the index (explicit) without considering the information leakage in accessing the file (implicit).

In this paper, a new dynamic searchable encryption sch-eme, called oblivious shuffle incidence matrix DSSE (OSM-DSSE), is proposed to access encrypted data obliviously under a single server. OSM-DSSE can hide both explicit and implicit search patterns in addition to the access pattern, contributing to a higher level of security. The contributions are as follows:

- This paper proposes a shuffling algorithm with Paillier encryption [23,24] to address the problem of access pattern leakage, which can shuffle the data in the incidence matrix to change the access path.
- This paper performs the search based on the group query and efficient 1-out-of-n OT protocol, ensuring the privacy of the server and the client. At the same time, random tokens can be generated to combine with the shuffling algorithm to hide the explicit search pattern.
- Since searching the same keyword always returns the same file set, exposing the response length is disclosed (implicit search pattern). This paper utilizes differential privacy based on the random response to hide the response length, so that the search pattern can be completely hidden.

1.1 Related work

Song et al. [1] first proposed the concept of static searchable encryption, making search encrypted data possible. Curtmola et al. [3] proposed the concept of adaptive security and proposed the first scheme with optimal search complexity $\mathrm{O}\left(\#\mathrm{DB}\left[\mathrm{w}\right]\right)$, where $\#\mathrm{DB}\left[\mathrm{w}\right]$ is the number of documents containing the keyword w. Many improvements have been made in their subsequent work [25,27,26]. Chase and Kamara proposed structured encryption to support queries on arbitrary-data-structure. Kamara et al. [4] proposed the concept of dynamic searchable encryption, making searchable encryption no longer limited to static operations. Although subsequent research efforts focused on effectiveness [17,28,29], dynamics [10,30,31], localization [32,

33], security [34–36], and complex functions [37–39], they still suffer from leaking some important information. Attackers can use these leakages to attack and recover data and cause more serious information leakages.

At present, some solutions have been proposed to deal with these leakages and attacks, but these researches primarily focused on forward-secure and backward-secure properties. Forward-secure refers to the ability to break the linkability of newly added data and query keywords; backward-secure means that the server is no longer able to match and retrieve the deleted data. Stefanov et al. [10] proposed the first solution to support forward-secure property, but it exhibits a linear time complexity for the search. Bost et al. [11] proposed a scheme relying on primitives such as constrained pseudorandom functions and puncturable encryption to achieve fine control of the opponent's power, preventing the adversary from evaluating functions on selected inputs, or decrypting specific ciphertexts for forward and backward security. Sun et al. [12] proposed the first practical, non-interactive backward-secure SSE scheme using symmetric punctured encryption. However, the forward-secure and backward-secure methods mainly aim at the information leakage in the update phase, without considering the information leakage of the access pattern and the search pattern. As a result, the problem of information leakage was not completely solved.

The oblivious random access machine (ORAM) can hide the access pattern by confusing each access process to make it indistinguishable from random access. The access pattern refers to the sequence of operations and memory addresses. The ORAM was first proposed by Goldreich et al. [13] to ensure that any data block in memory did not permanently reside at a physical address and that two accesses are unrelated. Goldreich et al. also proposed an ORAM model, giving a square root (Square-Root) and a layered solution. Zhang et al. [14] proposed a method based on ORAM access pattern protection in the cloud storage environment. Garg et al. [16] proposed a TWORAM scheme that reduces the client storage overhead while hides the file access pattern with ORAM. However, the researches have shown that using ORAM to eliminate information leaks leads to high overhead and low execution efficiency [40–42].

The rest of this paper is organized as follows. The preliminaries are presented in Section 2. The overview of the proposed scheme is described in Section 3. The detailed description of the algorithms is provided in Section 4. The security analysis is provided in Section 5. The experiment and analysis are provided in Section 6. The conclusion is in Section 7.

2 Preliminaries

In this section, some preliminaries used in the following sections are introduced. In this paper, $F = (f_1, f_2, ..., f_n)$ denotes a collection of n files, and $W = (w_1, w_2, ..., w_m)$ denotes a set of all possible keywords, where W_i represents the subset of W. Each file f_i associates with a unique keyword list $W_i \in W$.

2.1 Symmetric encryption

The symmetric key encryption scheme is represented as a tuple $\varepsilon(Gen, Enc, Dec)$ presenting $IND - CPA$ encryption scheme. It is used to encrypt and decrypt the documents, which usually consists of three polynomial-time algorithms:

- $K_F \leftarrow \varepsilon.Gen(1^\kappa)$: Key generation algorithm is a probabilistic algorithm. It accepts the input of a security parameter κ and outputs the key K_F for the file collection.
- $c \leftarrow \varepsilon.Enc_K(m)$: Encryption algorithm is a probabilistic algorithm. It accepts the inputs of a key K_F and a message m, and outputs the ciphertext c.
- $m \leftarrow \varepsilon.Dec_K(c)$: Decryption algorithm is a deterministic algorithm. It accepts the inputs of a key K_F and a ciphertext c, and outputs the message m.

2.2 Dynamic Symmetric Searchable encryption

A dynamic SSE scheme $DSSE = (KeyGen, BuilIndex, SrchToken, UpdToken, Search, Update, \varepsilon)$ is a tuple that consists of six polynomial-time algorithms and a symmetric key encryption scheme:

- $(K) \leftarrow KeyGen\,(1^\kappa)$ is a probabilistic key generation algorithm for the client to initialize. It takes the security parameter κ as input and outputs the key $K = (K_I, K_F)$, where K_I and $K_F \leftarrow \varepsilon.Gen(1^\kappa)$ are for the secure index and the file collections, respectively.
- $(I) \leftarrow BuildIndex\,(K_I, (F, W))$ is a probabilistic algorithm for the client to build a secure index. It takes the key K_I and a file collection F along with a keyword list W_i as inputs, and outputs a secure incidence I.
- $(c) \leftarrow \varepsilon.Enc\,(K_F, F)$ is a probabilistic algorithm for the client to encrypt the file collection. It takes the key K_F and file F as inputs, and outputs the ciphertext c. A ciphertext collection $C = (c_1, c_2, ..., c_n)$ is encrypted one by one.

- $(\tau_s) \leftarrow SrchToken(K_I, w)$ is a possibly probabilistic algorithm for the client to generate a search token. It takes the key K_I and a keyword w as inputs, and outputs a search token τ.
- $(R) \leftarrow Search(I, \tau)$ is a deterministic algorithm for the server. It takes the secure index I and search token τ as inputs, and outputs the search result R.
- $(\tau_u, c_f) \leftarrow UpdToken(K, f)$ is a possibly probabilistic algorithm for the client to generate an updated token. It takes the key K and a file f as inputs, and outputs an updated token τ_u.
- $(I', C') \leftarrow Update(I, c, \tau_u)$ is a deterministic algorithm for the server. It takes the secure index I, a ciphertext c, and an updated token τ_u as inputs, and outputs the new secure index I' and new ciphertext collection C'.
- $(F) \leftarrow \varepsilon.Dec(K_F, c)$ is a deterministic algorithm for the client to decrypt a ciphertext. It takes the key K_F, and a ciphertext c as inputs, and outputs a file F.

2.3 Paillier encryption

Paillier encryption [23], a public-key cryptosy-stem based on composite degree residuosity class, is an additive homomorphic encryption (AHE) scheme that provides semantic security. It is a tuple that consists of three algorithms $PE = (Gen, Enc, Dec)$:

- $(pk, sk) \leftarrow Gen(p, q)$: Key generation algorithm. It takes two large prime numbers p, q as input. Let $n = pq$ and $\lambda = lcm(p-1)(q-1)$, then select a $g \in Z_{n^2}^+$ with uniform probability, and $\mu = \left(L\left(g^\lambda \bmod n^2\right)\right)^{-1}$. It is noted that L is defined as $L(x) = \frac{x-1}{n}$, the public key is $pk = (n, g)$, and the private key is $sk = (\lambda, \mu)$.
- $c \leftarrow Enc_{pk}(m)$: Encryption algorithm. For the message $m \in Z_n^+$, a $r \in Z_N^+$ is selected with uniform probability, then it outputs the ciphertext $c = r^n g^m \bmod n^2$.
- $m \leftarrow Dec_{sk}(c)$: Decryption algorithm. For the ciphertext c, $m = D(c) = L\left(c^\lambda \bmod n^2\right) \cdot \mu \bmod n$ is decrypted, where $L(x) = \frac{x-1}{n}$.

2.4 Oblivious transfer protocol

The OT protocol [43] transforms information in a fuzzy way to effectively protect the privacy of the parties, i.e., the sender and the receiver. In this paper, the 1-out-of-n OT protocol is applied to the search result. The server and the client are the sender (S) and the receiver (R), respectively.

The efficient 1-out-of-n OT protocol is as follows:
Setup phase:

- System parameters: $(g, h, G(p))$, where p is a large public prime and g, h is the generator of z_p^* with $\log_g h$ unknown to anyone.
- Server's input: encrypted messages $\varepsilon(m_1), \varepsilon(m_2), ..., \varepsilon(m_n)$, where $\varepsilon(\cdot)$ indicates a symmetric encryption.
- Client's input: σ, where $1 \leq \sigma \leq n$.

Obliviously transform phase:

- The client randomly selects $r\,(r < p)$ to calculate $y = g^r h^\sigma \bmod p$ and sends it to the server.
- the server calculates $(\alpha_1, c_1), ..., (\alpha_n, c_n)$ and sends it to the client, where $\alpha_j = g^{k_j} \bmod p$, $c_j = \varepsilon(m_j)\left(y/h^j\right)^{k_j} \bmod p$, $\left(k_j \in_R Z_p^*, j = 1, ..., n\right)$.
- The client decrypts $\varepsilon(m_\sigma) = c_\sigma / \alpha_\sigma$ and obtains the symmetrically encrypted ciphertext.

The OT protocol has the following three basic properties:

Correctness: If the S and the R follow the protocol step by step, the R only obtains the selected message m_σ $(1 \leq \sigma \leq n)$

Sender's privacy: After S and R execute this protocol, R does not know any messages except its choice. That is, the ciphertexts are computationally indistinguishable for R.

Receiver's privacy: After S and R execute this protocol, S does not know which message the R has obtained. That is, the choice of R is computationally indistinguishable for S.

2.5 Random response

Random response [44] is a disturbance mechanism of differential privacy protection. It protects the privacy of the original data through the uncertainty of the response to sensitive issues. It often adds noise by tossing the coin. The question is formalized: 1. I have a sensitive attribute A (yes); 2. I have no sensitive attribute A (No). At this point, a coin is tossed. If the head is obtained, the answer is "yes". Otherwise, the coin is tossed again. If the head is obtained, the answer is "yes"; otherwise, the answer is "no". Due to the reasonable denial of "yes" and "no", the random response is used to protect privacy. The algorithm is formalized as $b' \leftarrow RandomResb, p, q$, where b represents the real answer, and b' is the output of a random response.

Suppose the value n needs to be perturbed randomly, and each respondent answers the question once. The number of people who answered "yes" is n_1, and

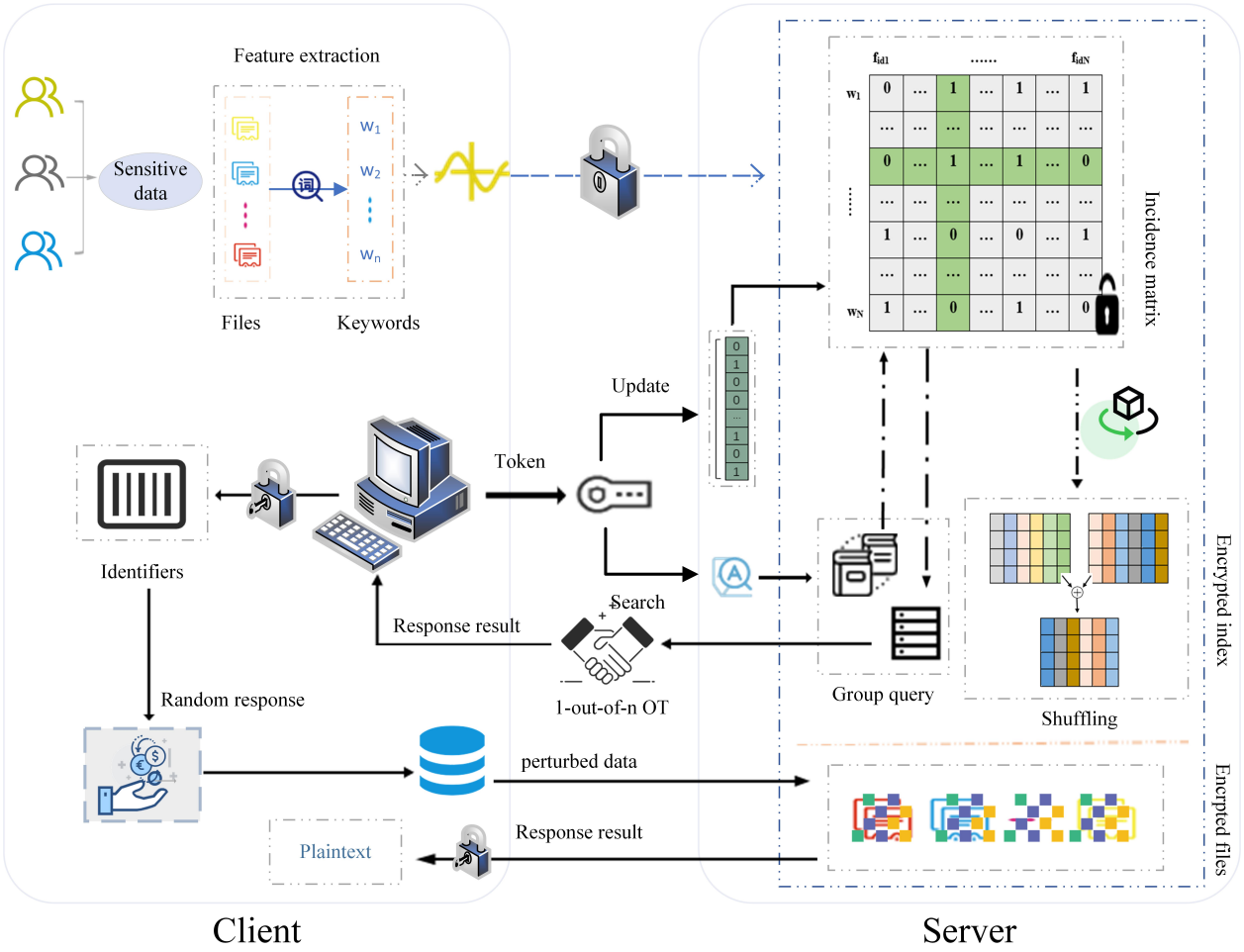


Fig. 1 The OSM-DSSE system model

the number of people who answered "no" is $n-n_1$. According to the statistics, the proportion of users who answered "yes" and "no" is as follows.

$$\begin{cases} \Pr\left(X =' Y'\right) = pq + (1-p)(1-q) \\ \Pr\left(X =' N'\right) = (1-p)q + p(1-q) \end{cases} \tag{1}$$

where, X represents the event, p represents the probability of head on the first coin, and q represents the probability of head on the second coin.

3 Overview of OSM-DSSE Scheme

3.1 System model

Our system utilizes the client-server model (refer to Fig. 1). The client extracts the keywords of the file and constructs an incidence matrix between the keywords and the file, encrypts the incidence matrix and the file, and sends them to the server. The client issues search and update requests to the server. The server stores the encrypted incidence matrix and responses to the client's search and update requests. Note that we consider a semi-honest (honest but curious) server. During the access, even though data files are encrypted, the cloud server may try to derive other sensitive information from users' search requests. Thus, although the server can faithfully follow the protocol, it can learn information.

3.2 System goal

Our goal is to effectively perform a privacy-protected keyword search and file update on an encrypted cloud databa-se. The main objectives of this system are as follows:

- Hide the access pattern
  We utilize Paillier encryption to shuffle the incidence matrix. This algorithm can randomize the position of keywords in the incidence matrix, confuse access paths, and hide access patterns.
- Hide the search pattern

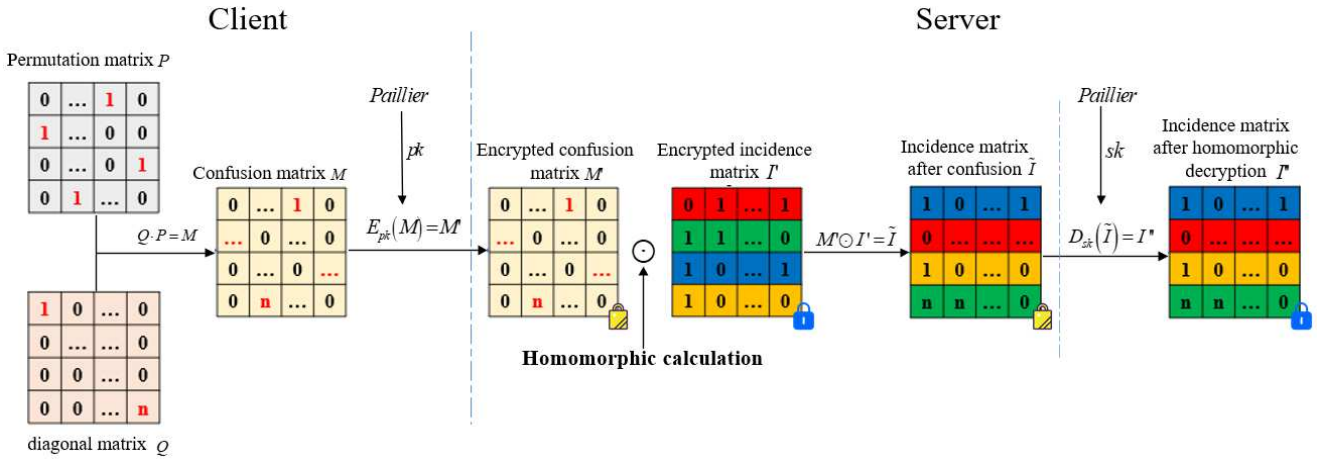


Fig. 2 Shuffling process

(1) Based on the group query, the server utilizes the two-level map to obtain the target data block containing multiple pieces of data. And it executes an efficient 1-out-of-n OT protocol with the client to obtain the target. In this process, the 1-out-of-n OT protocol makes the server unable to distinguish which keyword the client is searching for. The client also does not know the server's other messages except for the searched keyword. This protocol protects the privacy of the client and server simultaneously. Besides, the shuffling is performed after each search, which makes the row position of the keyword in the incidence matrix change, and it also can convert the deterministic token into a random token (explicit search pattern). The adversary cannot launch an attack by analyzing the search frequency.

(2) If the client searches for the same keyword, it always returns the same size file. The adversary can launch an attack by analyzing the response length. Th-erefore, this paper utilizes a differential privacy strategy based on random response to hide the response length (implicit search pattern).

3.3 System overview

The design goal of this system is to hide the search pattern and the access pattern of the searchable encryption scheme. Search and update are two core operations in the system.

3.3.1 Hiding the access pattern

The access pattern refers to the user's access path, where a repeated query is easily identified by the same access path. When the client sends a search request to the server, repeated searches also lead to the disclosure of the access pattern. The attacker can track the access path to obtain the query keyword and the information of the incidence matrix.

It is challenging to hide the access pattern while significantly reduce the computational and communicational cost of the searchable encryption scheme. The existing schemes [20,42] usually use the "fetch-decrypt-reencrypt-upload" strategy to hide the access pattern, but it causes high communication and computation overhead. The proposed scheme only uploads the confusion matrix to the server, and the server performs homomorphic calculation between the confusion matrix and the incidence matrix. The shuffling process is divided into two stages: shuffling and homomorphic decryption, which is shown in Fig. 2, it notes that the yellow lock indicates homomorphic encryption, and the blue lock indicates the symmetric encryption.

The specific procedure of shuffling is as follows. On the left in Fig. 2, the client calculates the confusion matrix based on the permutation matrix and the diagonal matrix, and the confusion matrix is encrypted with the Paillier pk. Here, some formulas are given to facilitate the calculation of the confusion matrix.

(1) Matrix-based data shuffling: Given a data sequence $B = (B_1, ..., B_n)$ and a $n \times n$ permutation matrix π, the position of the data block is changed by $B \cdot \pi$. For example: the blocks in $B = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$ can be changed with the permutation matrix $\pi = \begin{pmatrix} 01 \\ 10 \end{pmatrix}$:

$$\pi \cdot B = \begin{pmatrix} 01 \\ 10 \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} B_2 \\ B_1 \end{pmatrix} \tag{2}$$

(2) Matrix-based data scaling: Give a data sequence $B = (B_1, ..., B_n)$ and a $n \times n$ diagonal matrix C. The

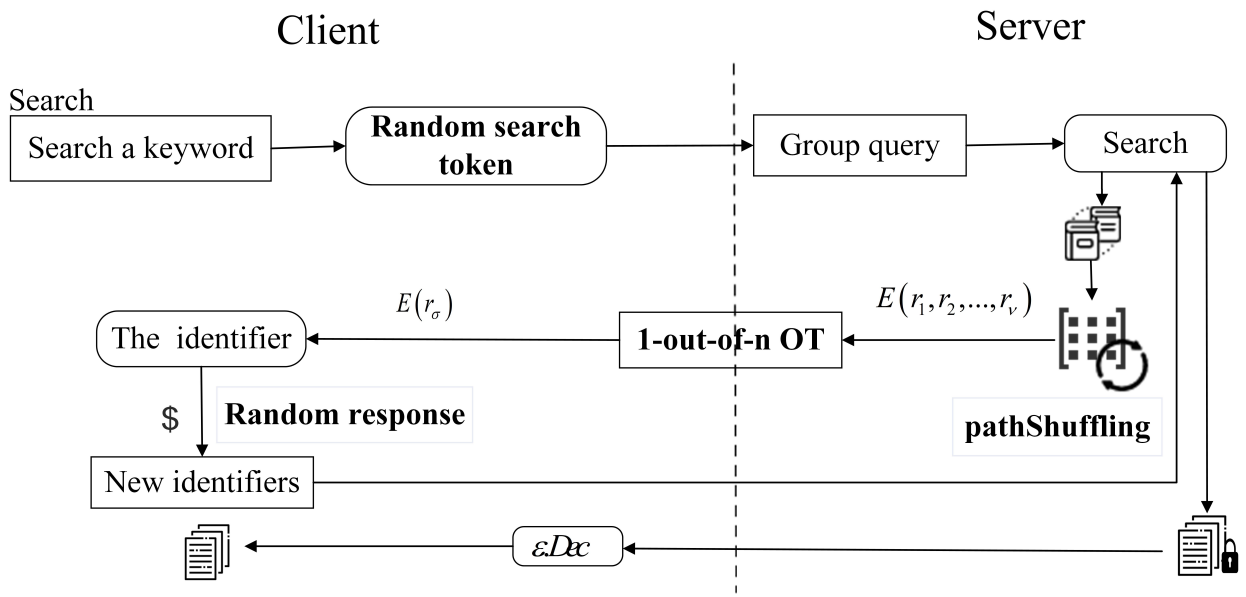


Fig. 3 Hidden search pattern diagram

matrix multiplication $C \cdot B$ can scale B with C. For example: given $B = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$ and scaling matrix $C = \begin{pmatrix} 20 \\ 03 \end{pmatrix}$, it can be obtained:

$$C \cdot B = \begin{pmatrix} 20 \\ 03 \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} 2B_1 \\ 3B_2 \end{pmatrix} \tag{3}$$

(3) Based on the formulas (2) (3), the confusion matrix can be obtained:

$$M = Q \cdot P \tag{4}$$

(4) The client encrypts the confusion matrix with the Paillier public-key pk:

$$M' \leftarrow E_{sk}(M) \tag{5}$$

On the right in Fig. 2, the server performs the homomorphic calculation between the encrypted confusion matrix and the incidence matrix to obtain the shuffled incidence matrix.

$$\widetilde{I} \leftarrow M' \odot I' \tag{6}$$

The row position in the incidence matrix is changed when the shuffling phase is over. Homomorphic decryption is performed subsequently. The incidence matrix on the server is encrypted with homomorphic encryption after shuffling. It can be seen from the property of Paillier encryption that two parties involved in the calculation are homomorphic encryption and non-homomorphic encryption. Therefore, to facilitate the next shuffle operation, the server needs to decrypt the incidence matrix with sk of the Paillier before the next search is performed. It should be noted that the client generates different public/private key pairs (pk, sk) of Paillier to resist malicious attacks by the server.

The server performs homomorphic decryption with sk to get the symmetric encrypted incidence matrix.

$$I'' \leftarrow D_{pk}\left(\widetilde{I}\right) \tag{7}$$

3.3.2 Hiding the search pattern

Simon et al. [9] pointed out that the search pattern can be divided into the explicit and the implicit. The explicit means searching for the same keyword always generates the same deterministic token, while the implicit means searching for the same keyword always returns the result set of the same size. Attackers can perform query recovery attacks using the query volume and the frequency leakages. Therefore, both the explicit and the implicit search patterns are hidden in this paper. The diagram of the hidden search pattern is shown in Fig. 3.

The group query is performed in this paper. It should be noted that if similar semantic keywords are grouped, the adversary can use similar semantics to infer the relationship between the keywords, causing a partial privacy leakage. So, the secure index is constructed where the location of keywords and files is determined by the pseudo-random function to ensure the randomness of the storage location. In this case, the attacker cannot infer the relationship between the keywords in the subsequent group queries.

For search, the two-level map and the efficient 1-out-of-n OT protocol are utilized to hide the search pattern.

However, the adversary can still guess the search target by the response length. Therefore, the random response is used to further hide the response pattern (implicit). Also, after the search, the server performs the shuffling operation (shown in Fig. 2) to change the row position of the incidence matrix. Since the search token is related to the position of the keyword, the shuffling also converts the deterministic search token into a random search token.

The client utilizes the pseudo-random function and hash to randomly locate the keywords before performing the group query. The search process is as follows. The client obtains the line number of the search keyword according to the dictionary D and calculates the block number l to which the keyword belongs. Then, the client's selection σ is combined to generate a search token and send it to the server. Once the server receives the search token, it retrieves the two-level map Ω to obtain a row number group of size ν according to the search token, and then retrieves the incidence matrix according to the row number group to obtain a data block of size ν. After that, the server and the client execute an efficient 1-out-of-n OT protocol. The server returns the search results to the client for decryption. The client obtains the file identifier set containing the searched keyword. Then, to hide the response length (implicit), the random response is utilized to randomly perturb the set of file identifiers, and the file set containing virtual items from the server is returned.

After the search, the incidence matrix needs to be shuffled with the shuffling algorithm to generate the random search token.

4 Details of the Proposed OSM-DSSE Scheme

4.1 Data structure

1. Incidence matrix. The incidence matrix is used to build an encrypted incidence, and its elements represent the correspondence between keywords and files. Specifically, the row indicates the keyword w, and the column indicates the file id. If the keyword of the i-th line appears in the file of the j-th column, $I[i,j] = 1$; otherwise $I[i,j] = 0$. Search and update operations access a row or column of the incidence matrix. I is encrypted with the IND-CPA tuple $\varepsilon(Gen, Enc, Dec)$.
2. Two hash tables T_f and T_w. The location of files and keywords in I is determined when constructing the incidence matrix for search or update operation.

   (1) File hash table. $T_f\left(s_{id_j}, \langle y_j, st\rangle\right)$, where $s_{id_j} \leftarrow G_{k_2}(id_j)$ is the key and G is a pseudo-random function. s_{id_i} indicates a file with an identifier id_i, and the column index $y_j \in \{1, ..., n\}$ is equal to the position of s_{id_j} in T_f. st means status, $st = 1$ means add while $st = 0$ means delete.

   (2) Keyword hash table. $T_w(s_{w_i}, x_i)$, where $s_{w_i} \leftarrow G_{k_2}(w_i)$ is the key, and w_i indicates the keyword. The row index $x_i \in \{1, ..., m\}$ is equal to the position of s_{w_i} in T_w.
3. Dictionary D. The dictionary stores key-value pairs $D[w] = x$, recording keywords and the corresponding line numbers.
4. Two-level map $\Omega(M_w, A)$. It consists of an address map table and an array. For the address map table $M_w(l, i)$, the server searches the array index i based on the block number l, where $A[i]$ represents the value corresponding to the position of $i + 1$. This value is used to determine the row of the search keyword.

   The two-level map is constructed through the following three steps:

   a) The dictionary D is partitioned into ν-blocks $I_1, ..., I_t$ and I_t is padded up to ν elements if necessary.

   b) The block number is taken as the key of the M_w, and the value corresponding to the key is the starting incidence of each data block in the array A.

   c) Store the row number of the incidence matrix in the array A.
5. Address map table $M_f(s_{id}, j)$. For the update, the server determines the column j where the update file is located according to the update token.

4.2 Construction of OSM-DSSE

In this section, the OSM-DSSE scheme is defined, consisting of the four algorithms ($Setup, Search, pathShuffing, Update$) presented in the subsections below.

4.2.1 Setup

- $(K, I', C) \leftarrow Init(\kappa, \sigma, F)$: various parameters are input to obtain the public parameters.

Firstly, the client generates public parameters by $DSSE.KeyGen$. These parameters include the symmetric key K_F to encrypt files and the key K_I to encrypt index.

Secondly, the client constructs a secure random incidence matrix by $DSSE.BuildIndex$. As shown in Algorithm 1, the client generates a random key k_2. The client extracts the keyword set $W = (w_1, ..., w_m)$ from the file set $F = \{f_1, ..., f_n\}$ (each file has a unique identifier $(id_1, ..., id_n)$). The positions of each keyword and

file in the incidence matrix are determined by the pseudorandom function G and the hash tables T_f and T_w.

Then, the client divides the keyword set W into data blocks of size ν. The last data block is padded up to ν elements if necessary, and it is numbered as $(1, 2, ..., \lceil m/\nu \rceil)$. The client constructs a two-level map $\Omega(M_w, A)$ and encrypts the key of M_w. Simultaneously, the client constructs the dictionary D according to the $T_w(s_{w_i}, x_i)$ and the M_f according to the $T_f\left(s_{id_j}, \langle y_j, st\rangle\right)$.

Lastly, the client encrypts the file by $\varepsilon.Enc$, and sends the secure random incidence matrix I', encrypted file C, two-level map Ω, and address map table M_f to the server. Meanwhile, the client saves δ locally.

Algorithm 1: *BuildIndex*

Input: files and keywords collection (F, W)
Output: secure random incidence matrix I

$k_2 \xleftarrow{\$} \{0,1\}^{\kappa}$
Extract $(w_1, ..., w_m)$ from $F = \{f_{id_1}, ..., f_{id_n}\}$
for $i \in (1, ..., m)$ **do**
  $s_{w_i} \leftarrow G_{k_2}(w_i), x_i \leftarrow T_w(s_{w_i})$
  for $j \in (1, ..., n)$ **do**
    if w_i *appears in* f_{id_j} **then**
      $s_{id_j} \leftarrow G_{k_2}(id_j), y_j \leftarrow T_f\left(s_{id_j}\right)$
      $I[i,j] \leftarrow 1$
    end
  end
end
$I'[i,j] \leftarrow I[i,j] \oplus H(K_I||r_i)$

4.2.2 Search

- $R \leftarrow Srch(\tau)$: Input search token and get response results.

The client generates the search token $\tau \leftarrow (l||k_3, y)$ by $DSSE.SrchToken$. Where $(l||k_3)$ is the encrypted block number and y is the client's selection that contains search target. The client obtains the row number x_i by $x_i \leftarrow D[w_i]$ and calculates $y = g^r h^{\sigma} \bmod p$, where $r\,(r < p)$ is a random value.

The server performs group queries according to the search token as follows:

Firstly, the server parses $\tau \leftarrow (l||k_3, y)$ and queries the $M_w(l||k_3, i)$ of $\Omega(M_w, A)$ to obtain the starting position of the keyword in the array A, and then sequentially searches A to obtain the $row = (r_i, ..., r_{i+v})$. Secondly, the server searches the encrypted incidence matrix according to $row = (r_i, ..., r_{i+v})$ and obtains a data block $B = (b_1, .., b_\nu)$ of size ν, which is symmetrically encrypted.

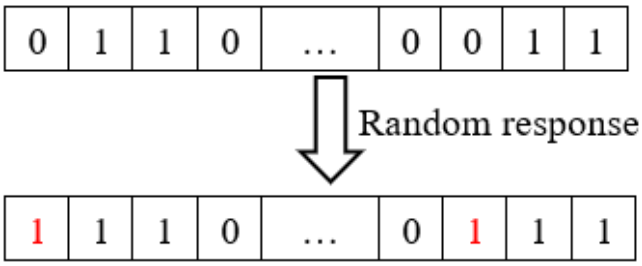


Fig. 4 Random response

Then, for this result set, the server and the client execute the efficient 1-out-of-n OT protocol, where the value of n is to ν. The server calculates $((\alpha_1, c_1), ..., (\alpha_\nu, c_\nu))$, where $\alpha_j = g^{k_j} \bmod p$, $\left(k_j \in_R Z_p^*, j = 1, ..., \nu\right)$, α_ν is the auxiliary parameter, and c_ν is a ciphertext sent by the server to the client.

Finally, the client decrypts the ciphertext. The client performs the first decryption $\varepsilon(m_\sigma) = c_\sigma/\alpha_\sigma$ to obtain the result of symmetric encryption. Then, the client performs the second decryption with the initial row number r_i and the key K_I to obtain the file identifiers containing the keyword to be queried, i.e., $\Upsilon_w \leftarrow \varepsilon(m_\sigma) \oplus H(K_I||r_i)$, where $\Upsilon_w = (0,1)^n$ indicates the result vector. At this time, the random response is used to fill the result vector randomly and perturb the data (refer to Fig. 4), shown in Algorithm 2. The size of the result file set returned each time is different.

Algorithm 2: $Obfuscate(\Upsilon, p, q)$:

Input: identifiers collection Υ, probabilities p and q
Output: a new identifiers collection Υ'

$c = 0$
for $j \in (1, ..., n)$ **do**
  if $\Upsilon[j] == 1$ **then**
    $c++$
  end
end
$p = \frac{c}{n}$ //The probability of 1 in the result vector
for $j \in (1, ..., n)$ **do**
  if $\Upsilon[j] == 1$ **then**
    $b' == 1$
  else
    $\Upsilon[j] == 0$
  end
  $b' \leftarrow RandomRes(0, p, q)$
end
if $b' == 1$ **then**
  $\Upsilon[j] \leftarrow b'$
end
return Υ'

The server returns the file sets containing dummy items to the client based on the result vector after perturbation. After getting the encrypted data, the client

decrypts the data using the key K_F to obtain the data that satisfies the query.

4.2.3 PathShuffling

- $I'' \leftarrow PathShuffling\,(I')$: Input the incidence matrix to be shuffled, and output the new incidence matrix after shuffling.

After the search, the server executes the path shuffling, as shown in Fig. 2. The client constructs and encrypts the confusion matrix with Paillier, then uploads it to the server. The server performs the homomorphic calculation. This process can change the access path to hide the search pattern and access pattern.

First, the client constructs the permutation matrix P and the diagonal matrix Q. The dot product operation is performed on P and Q to form the confusion matrix M. The realization of P is as follows. $\pi_i = i'$ represents a random permutation function, where $i = 1, ..., 2n$. This function generates a permutation π by randomly and uniformly selected items from the set $\{1, ..., 2n\}$. Let $p_{i,j}$ denotes the value of the i-th row and the j-th column in the P, and P can be expressed as $p_{i,j} = \begin{cases} 1 i = \pi(j) \\ 0 \end{cases}$. The diagonal matrix Q is a randomly generated $m \times m$-dimensional invertible matrix, which contains m diagonal numbers, i.e.,$Q = \begin{bmatrix} q_1 \; 0 \; 0 \\ 0 \; ... \; 0 \\ 0 \; 0 \; q_m \end{bmatrix}$.

The client generates a public and private key pair (pk, sk) by $PE.Gen$, and encrypts the confusion matrix M with pk, i.e., $M' = PE.Enc_{pk}\,(M)$. Then, the client sends M' to the server. The server performs homomorphic calculation between the M' and I' to obtain the shuffled incidence matrix $\widetilde{I}$, namely $\widetilde{I} \leftarrow M' \odot I'$. Since the Paillier encryption satisfies semantic security, the same plaintext can generate different ciphertexts. The result is homomorphical encryption after performing the homomorphic calculation. However, this calculation is not conducive to the next data shuffling. So, the client sends the sk to the server for homomorphic decryption before the next search, i.e., $I'' \leftarrow PE.Dec_{sk}\left(\widetilde{I}\right)$, where I'' represents the result of symmetric encryption. After shuffling, the position of the keyword in the incidence matrix will be changed. To correctly obtain the row number of the next keyword for search, the client needs to update the dictionary D. Moreover, the client should generate the different public and private key pairs to ensure the server cannot decrypt the confusion matrix.

4.2.4 Update

- $(C', I') \leftarrow Upda\,(\tau_U)$: Input the update token, and output the updated encrypted file and incidence matrix.

The update operation needs an interaction between the client and the server, and it contains add and delete operations. An update token is generated by $DSSE.Upd\;Token$ to perform updates by the server. It should be noted that the proposed solution will not reveal the update type, since both the add and delete operations are written back to the server.

For add operation. The client confirms the column j is to be added by T_f and sets the status value to 1. The client extracts the keywords of the file and constructs a column matrix $\overline{I}$ according to the T_w before it encrypts the file by $\varepsilon.Enc$. Then, the client sends (τ_a, c') to the server. The server utilizes the token to update the incidence matrix I', the address map table M_f and the ciphertext C.

For delete operation. The client confirms the column j is deleted by T_f and sets the status value to 0. The client constructs a column matrix $\overline{I}$ with all 0. Then, the client sends τ_d to the server. The server utilizes the token to update the incidence matrix I', the address map table M_f and the ciphertext C.

5 Security analysis

To prove the security of the above scheme, some definitions and theorems are given first.

Definition 1 The leakage function $L = (L_{stp}, L_{srch}, L_{upd})$ is defined as follows.

(1) $(N, ID, <|c_1|, ..., |c_n|>) \leftarrow L_{stp}\,(I', C)$: Input the encrypted incidence matrix I' and the encrypted file set C. Output the maximum value N of keywords and files, the file identifier $ID = \{id_1, ..., id_n\}$, and the encrypted file size $|c_{id_i}|$ $(1 \le i \le n)$.
(2) $(\tau) \leftarrow L_{srch}\,(I', Q)$: Input the encrypted incidence matrix I' and the query Q. Output the searched block number τ.
(3) $(j, |c_i|) \leftarrow L_{upd}\left(I', \overline{I}\,[j], c_j\right)$: Input the encrypted incidence matrix I', the encrypted column matrix $\overline{I}\,[j]$, and the encrypted file c_j. Output the column number j of the encrypted file and the size $|c_i|$ of the encrypted file.

Definition 2 The simulator $S = (SimStp, SimSrch, Sim, Upd)$ is defined as follows.

(1) $\left(I_{M_{Sim}}{}', C_{Sim}\right) \leftarrow SimStp\left(N, ID, <|c_1|, ..., |c_n|>\right)$: The simulator constructs an encrypted incidence matrix and an encrypted file with random values according to the information $(N, ID, <|c_1|, ..., |c_n|>)$ lea-ked by the leakage function $L_{stp}\left(I', C\right)$. Then, the encrypted incidence matrix and file are sent to the adversary A.

(2) $\tau^s_{b,k} \leftarrow SimSrch\left(l\right)$: The simulator randomly selects a keyword to simulate the search token $\tau^s_{b,k}$ according to the block number information l leaked by the leakage function $L_{srch}\left(I', D, Q\right)$. Then, the token is sent to the adversary A for the search.

(3) $\left(\tau^s_{f_{b,k}}, c_i\right) \leftarrow SimUpd\left(j, |c_i|\right)$: The simulator randomly selects a file based on the information $(j, |c_i|)$ leaked by the leakage function $L_{upd}\left(I_M{}', I_T, I[i], c_i\right)$ to simulate the update token $\tau^s_{f_{b,k}}$ and the ciphertext file c_i. Then, the token and the ciphertext file are sent to the adversary A for the update.

Definition 3 Local Differential Privacy. For an algorithm A, $Opt\left(A\right)$ denotes the set of all possible outputs of A. The algorithm A satisfies ε-local differential privacy ($\varepsilon \geq 0$) for any input x_1, x_2, it holds:

$$\Pr\left[A\left(x_1\right) = y\right] \leq e^{\varepsilon} \cdot \Pr\left[A\left(x_2\right) = y\right]$$

Where $y \in Opt\left(A\right)$ and ε represents the privacy budget $\varepsilon = \ln\left(1 + \frac{1-q}{pq}\right)$.

Corollary 1 *Given the perturbation probabilities p and q of random response, the proposed scheme satisfies ε-differ-ential privacy, where* $\varepsilon = \ln\left(1 + \frac{1-q}{pq}\right)$.

Proof. It is assumed:

$$\begin{cases} \Pr\left[b = 1\right] = p \\ \Pr\left[b = 0\right] = 1 - p \\ \Pr\left[b' = 1 | b = 0\right] = q - pq \\ \Pr\left[b' = 0 | b = 0\right] = 1 + pq - q \end{cases} \tag{8}$$

Where b is input, and b' is the output of random response.

According to the definition of differential privacy, the two inputs b_1, b_2 with only one different bit lead to the same output result of b'. Assuming b_1, b_2 differ in location i and location j, it holds

$$\frac{\Pr\left(b' | b_1\right)}{\Pr\left(b' | b_2\right)} = \frac{\left(q - pq\right)^{b'_i} \left(1 + pq - q\right)^{1 - b'_i} \times p^{1 - b'_j} \left(1 - p\right)^{b'_j}}{p^{1 - b'_i} \left(1 - p\right)^{b'_i} \times \left(q - pq\right)^{b'_j} \left(1 + pq - q\right)^{1 - b'_j}}$$

$$\leq \max\left\{1, \frac{\left(q - pq\right) p}{\left(1 - p\right)\left(1 + pq - q\right)}, \frac{\left(1 + pq - q\right)\left(1 - p\right)}{p\left(q - pq\right)}\right\}$$

$$= \frac{\left(1 + pq - q\right)\left(1 - p\right)}{p\left(q - pq\right)}$$

$$\Rightarrow \left(\frac{\left(1 + pq - q\right)}{pq}\right) = e^{\varepsilon}$$

$$\Rightarrow \varepsilon = \ln\left(1 + \frac{1 - q}{pq}\right)$$

Theorem 1 *Adaptive Semantic Security. Suppose that* $\Sigma_{OSM} = (Setup, Search, pathShuffling, Update)$ *is an interactive sch-eme based on an incidence matrix that hides the search pattern and the access pattern, and* $\lambda \in \mathrm{N}$ *is a security parameter. There is a leakage function* $L = (L_{stp}, L_{srch}, L_{upd})$ *for any PPT stateful adversary A that issues a polynomial query q, and there is a stateful simulator* $S = (SimStp, SimSrch, SimUpd)$ *so that:*

$$\left|\Pr\left[Real^{\Sigma_{OSM}}_{\mathrm{A}}\left(\lambda\right) = 1\right] - \Pr\left[Ideal^{\Sigma_{OSM}}_{\mathrm{A},S,L}\left(\lambda\right) = 1\right]\right| \leq negl\left(\lambda\right)$$

It can be proven that Σ_{OSM} *exhibits adaptive semantic security under L.*

Proof. For all PPT adversary A, the difference between the output probability of the $Real^{\Sigma_{OSM}}_{\mathrm{A}}\left(\lambda\right)$ and $Ideal^{\Sigma_{OSM}}_{\mathrm{A},S,L}\left(\lambda\right)$ given in Theorem 1 is a negligible value. A series of five games are defined to prove the security (refer to the Appendix for details). The first game is a real experiment, and the last game is an ideal experiment. Also, the success event of each game is defined, where $Game_i$ stands for the event that the opponent correctly guesses the challenge bit b, and the $\Pr\left[Game_i = 1\right]$ represents the probability of the succeeding adversary attacks. The security is proven by the progressive relationship of the related games, and the full proof is provided in the Appendix.

6 Experiment and analysis

6.1 Performance analysis

6.1.1 Storage overhead

Client storage: The client maintains two hash tables and a dictionary. Storage costs of the two hash tables are proportional to the number of keywords and files, i.e., $O\left(m\right)$ and $O\left(n\right)$, where m, n represents the number of keywords and files, respectively. The storage cost of the dictionary is proportional to the number of keywords, i.e., $O\left(m\right)$. Therefore, the total storage cost of the client is $O\left(m + n\right)$.

Server storage: The server maintains the incidence matrix I', a two-level map Ω, and an address map table M_f. The incidence matrix is a $m \times n$-dimensional matrix with a storage cost of $O(m * n)$. The two-level map Ω consists of two parts: an address map table M_w and an array A. Storage of M_w is proportional to the number of blocks. Assuming that the data are divided into t blocks, the storage cost of M_w is $O(t)$. The size of array A is related to the number of rows of the incidence matrix, and the storage cost is $O(m)$. The storage cost of M_f is related to the number of files, and the storage cost is $O(n)$. Therefore, the total storage cost of the server is $O(m \cdot n + m + n + t)$.

6.1.2 Communication overhead

In the setup phase, the client sends the encrypted incidence matrix and the encrypted file to the server. The communication overhead is $O(m * n + nc_i)$, where $m \times n$ is the size of encrypted incidence matrix and c_i is the size of each encrypted file.

In the search phase, the client sends the $m \times m$ confusion matrix to the server, and the communication overhead is $O(m^2)$. The server returns an encrypted data block of size ν with a communication overhead of $O(\nu)$.

In the update phase, the client sends the $m \times 1$ column matrix to the server, and the communication overhead is $O(m)$.

6.1.3 Computational overhead

Client computational overhead. The client mainly generates a permutation matrix, a diagonal matrix, and an encrypted confusion matrix. Both the permutation matrix and the diagonal matrix are $m \times m$ dimensions, so the confusion matrix is of dimension $m \times m$. In addition to the permutation matrix and the diagonal matrix, there are m pieces of data that are not 0. The remaining numbers are all 0, and the computational cost of generating 0 is negligible. So, the computational cost of generating the permutation matrix and the diagonal matrix is $\mathrm{O}(m)$.

Server computational overhead. The server needs to re-encrypt data of size ν and perform 2ν times of modular exponentiation operation when executes the 1-out-of-n OT protocol. The server mainly performs the homomorphic calculation between the confusion matrix and the incidence matrix. The size of the confusion matrix is $m \times m$, and the target matrix has m rows. So, the computational cost is $\mathrm{O}(m^3)$.

6.2 Experimental evaluation

6.2.1 Experiment preparation

The proposed OSM-DSSE scheme is evaluated in a real network environment and system setting. For search operation, a round of interaction is defined as $client -> server -> client$, which means that a search request is sent from the client to the server, and the data block is then downloaded from the server to the client . For update operation, a round of interaction is defined as $client -> server$, indicating that the client sends an update request to the server.

The hardware of the client and the server are configured as follows. The hardware configurations of client are Intel Core i5-8400 CPU @ 2.80 Hz, 16 GB RAM, 256 GB hard disk, and 1TB SSD. Besides, the client runs an operating system of Windows 10 64 bit. The hardware configurations of the server are 32 CPUs @ 2.70 GHz and 512 GB RAM. And the operating system of the server is CentOS 7.2 64-bit.

The Google sparse hash is used to realize the hash table T_f and T_w, and the hash tables are saved on the client. The file and the incidence matrix are pre-encrypted with the IND-CPA and sent to the server.

The online public dataset Enron [45] (mail dataset) is taken as the experiment dataset. The dataset contains data from approximately 150 users, and the corpus contains a total of about 500,000 messages. In the experiment, the emails of the 150 users are used. Since most of the emails are personal, they capture informal conversations between two individuals. Therefore, a stemming algorithm, namely the Porter Stemming Algorithm [46], is used to find each word's root in the document set and delete the most common words such as 'the', 'a', and 'from' to extract keyword sets from the corpus. For comparison, 300,000 files and 300,000 keywords are selected to construct an encrypted incidence matrix of different sizes (the largest incidence matrix has 9×10^{10} keyword-file pairs).

6.2.2 Experimental results

In the experiment, the performance of search and update operations of the proposed scheme is evaluated and compared with existing schemes.

The time for creating an incidence matrix of different sizes is evaluated to illustrate how the size of the dataset influences the construction time of incidence matrix. As shown in Fig. 5, the construction time is 10.114 s for an incidence matrix of $10^3 \times 10^3$, When the size of the incidence matrix exceeds $10^3 \times 10^3$, the time to construct the encrypted incidence matrix increases

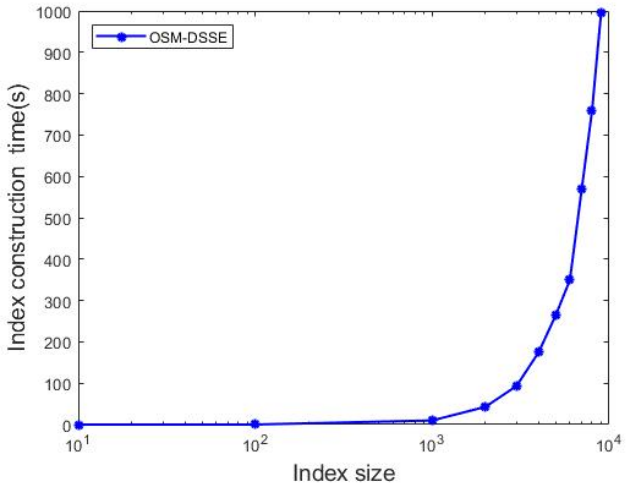


Fig. 5 The relation between index size and construction time

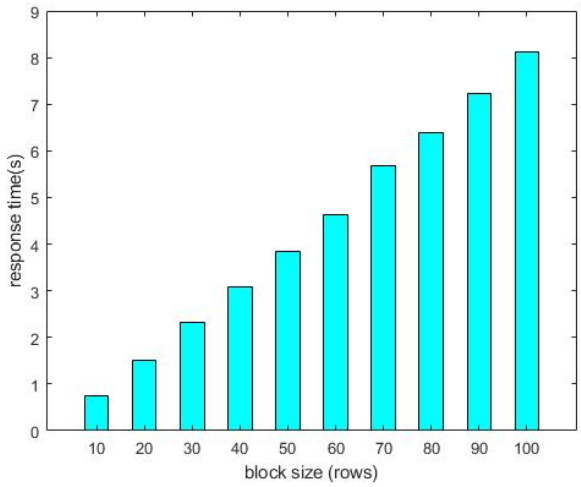


Fig. 7 The relation between response time and block size

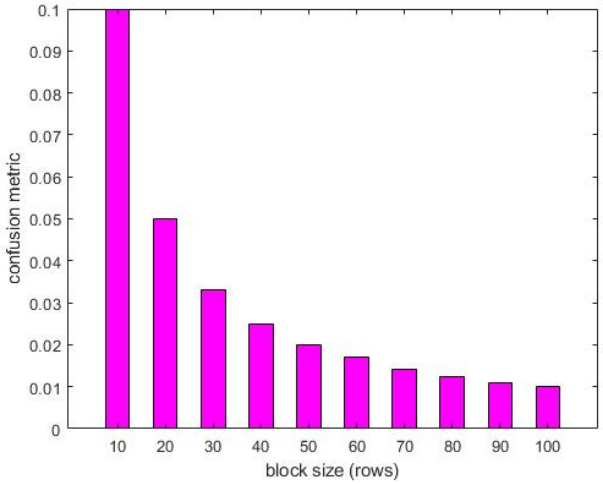


Fig. 6 The relation between confusion metric and block size

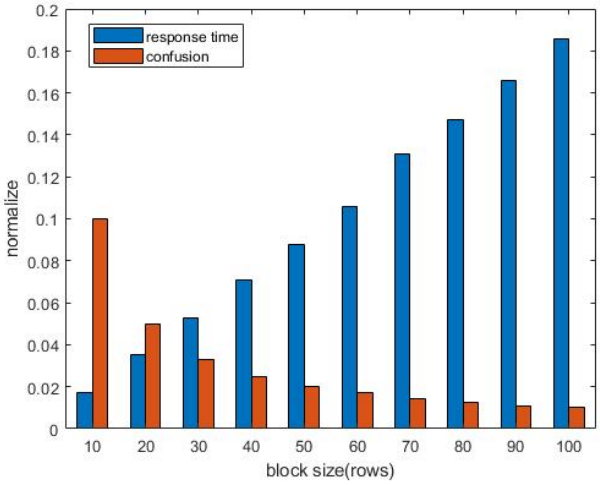


Fig. 8 The comparison between confusion and response time

rapidly. For example, it takes approximately 20 minutes to construct a $10^4 \times 10^4$ incidence matrix with 10^8 data. Since the incidence matrix is only constructed once during the setup phase, the relationship among the search, update time, and the size of the incidence matrix is mainly investigated.

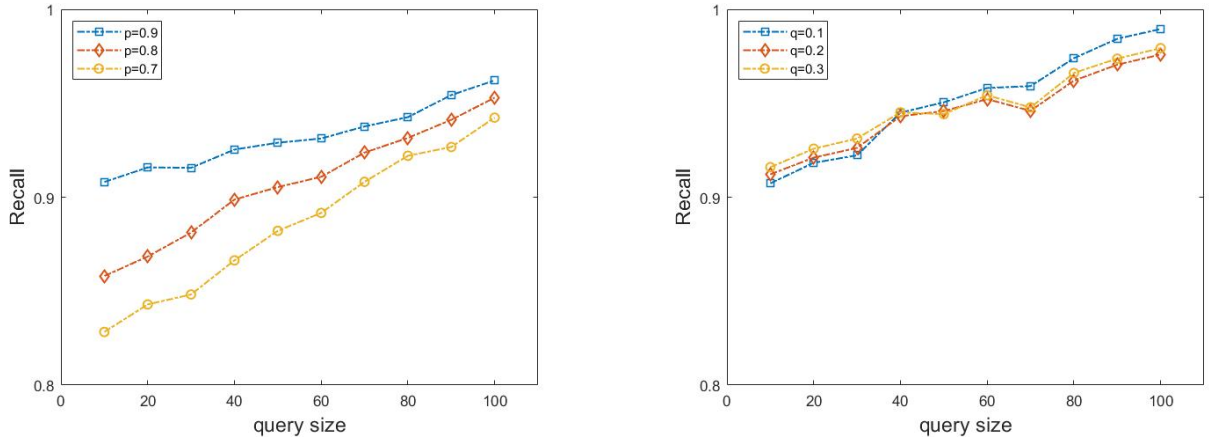
Then the size of the data block is evaluated. For search operation, the server returns a data block to the client. A data block is composed of multiple rows of the incidence matrix. The size of the data block not only affects the search performance but also influences the hiding effect of the search pattern. Therefore, it is essential to choose a suitable block size. Here, the confusion metric is defined.

Definition 4 The confusion metric Ψ is defined as the probability that the server guesses the target item when it performs a search request. The confusion metric is formalized as $\Psi = \frac{1}{\nu}$. where ν is the size of the data block.

For the data block size, a multiple of 10 rows in $[10, 100]$ is selected as the experimental data to evaluate the confusion metric. It can be seen from Fig. 6 that the probability of the server correctly guessing the target item decreases along with the size of the data block increases, which shows a better hiding effect of the search pattern. Also, the confusion curve tends to be flat when the size of the data block exceeds 30 rows.

The response time of different data block sizes is evaluated. As shown in Fig. 7, the response time is 0.76s, 1.51s, and 2.32s for data blocks with the size of 10 rows, 20 rows, and 30 rows, respectively.

To obtain the optimal block size that contributes to both good confusion metric and response time, the response time is normalized to a range of $[0, 1]$. Fig. 8



(a) Fix q=0.1 (b) Fix p=0.9

Fig. 9 The effect of returned result size on recall

Table 1 Comparison of our scheme and its counterparts

Scheme	Storage		Communication		Hide search pattern	Hide access pattern
	client	server	search	update		
$OSM-DSSE$	$O(m+n)$	$O(m \cdot n)$	$O(\nu)$	$O(m)$	√	√
$IM-DSSE$ [47]	$O(m+n)$	$O(m \cdot n)$	$O\left(\frac{n}{p}\right)$	$O(m)$	×	×
$ODSE$ [21]	$O((m+n)(\log n+1))$	$O(km \cdot n)$	$O(m+n)$	$O(m+n)$	×	√
$DOD-DSSE$ [20]	$O(n)$	$O(kn^2)$	$O(n)$	$O(n)$	√	√

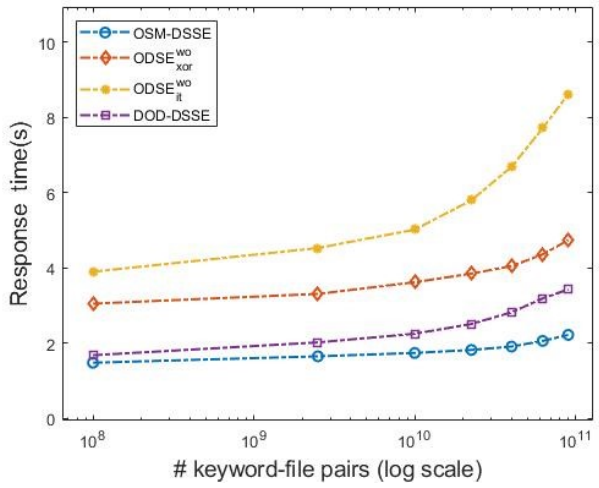


Fig. 10 The relation between search time and number of keyword-file pairs

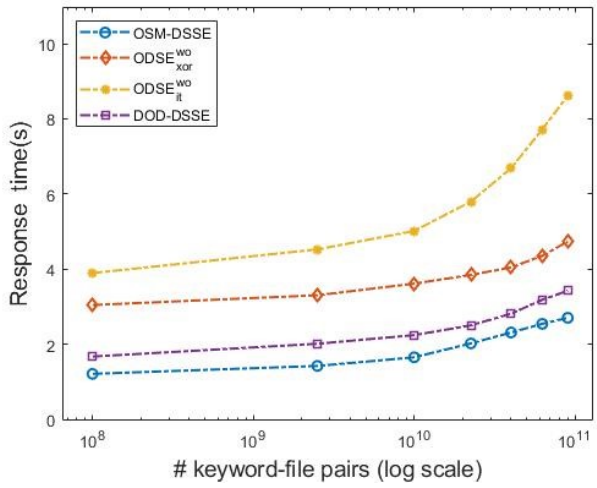


Fig. 11 The relation between update time and number of keyword-file pairs

shows the relationship of confusion and response time under different data block sizes, where the red bar and the blue bar represent the confusion and the response time, respectively. It can be seen that the difference between the confusion and the response time exhibits less variation for data blocks of 20 rows and 30 rows. Since the main purpose of this paper is to hide the search pattern, data block size of 30 is selected. Then, the search and update time of proposed scheme is compared with that of other schemes under the different sizes of the incidence matrix.

Recall. the recall $\xi = \frac{|\Upsilon|}{|\Upsilon'|}$ is defined, where Υ is the set of true results; Υ' is the result set after adding noise, and $||$ is the number of 1 in the result set. According to Algorithm 2, probability p represents the probability of 1 in the result set, and q represents the probability of adding noise 1 to the confusion result set. Therefore, the impact of different p and q on the recall is evaluated, and the result is shown in Fig. 9. It can be seen from Fig. 9 (a) that for a fixed q, the larger p usually corresponds to a higher recall. For example, when p=0.9 and q=0.1, the value of recall exceeds 0.9.

As shown in Table 1, the proposed scheme OSM-DSSE is compared with some exiting schemes in terms of storage overhead, communication overhead, and the ability to hide search and access patterns. The overhead of all schemes is measured on average. For the server-side storage, only the size of the encrypted incidence is considered. m and n respectively denote the maximum number of keywords and files. k represents the number of servers, ν represents the size of the data block, p represents the number of processors.

The IM-DSSE is a traditional DSSE scheme that leakages the search pattern and access pattern. The ODSE employs multi-server PIR and Write-Only ORAM to hide the access pattern. The DOD-DSSE leverages two non-colluding servers to realize the "fetch-reencrypt-swap" strategy, so that the data structure-access pattern can be hided. Compared with the above schemes, the proposed scheme not only achie-ves a low storage and communication overhead, but also hides the search pattern and access pattern.

Finally, the performance of search and update operations of the proposed scheme is compared with that of DOD-DSSE [20] and ODSE [21] under different incidence matrix sizes, and the results are shown in Fig. 10 and Fig. 11. The DOD-DSSE leverages two non-colluding servers and exploits the properties of an incidence matrix to avoid information leakages. The ODSE harnesses the Write-Only ORAM for update operation and multi-server PIR for search operation, achieving a low end-to-end delay and good Infor-mation-theoretic security. It sho-uld be noted that although Paillier is used to achieve shuffling in the proposed scheme, the shuffling operation is performed after the search, which does not affect the search performance. It can be seen from Fig. 10 that the search time of OSM-DSSE, DOD-DSSE, $ODSE^{wo}_{xor}$, 10^8, and $ODSE^{wo}_{it}$ is 1.45 s, 1.68 s, 3.05 s and 3.91 s respectively for the keyword-document pair with the size of 10^8. When the size of the keyword-document pair increases to 9×10^{10}, the search time of the $ODSE^{wo}_{it}$ is approximately 4x compared to that of the OSM-DSSE.

For the update operation, it can be seen from Fig. 11 that the update time of OSM-DSSE, DOD-DSSE, $ODSE^{wo}_{xor}$ and $ODSE^{wo}_{it}$ is 1.22 s, 1.68 s, 3.05 s and 3.91 s respectively for the keyword-document pair with the size of 10^8. When the size of the keyword-document pair increases to 9×10^{10}, the update time of the $ODSE^{wo}_{it}$ is approximately 3x compared to that of the OSM-DSSE.

7 Conclusion

This article proposes a searchable encryption scheme na-med OSM-DSSE to hide the search and access patterns. An effective shuffling algorithm based on Paillier is proposed to shuffle the incidence matrix, so that the position of the row in the incidence matrix is changed. This scheme combines the 1-out-of-n OT protocol and the differential privacy strategy based on random response to realize random data access. Besides, the security of the proposed scheme is formally analyzed, showing that the proposed scheme provides an adaptive semantic security that can against selective adversaries. Furthermore, the OSM-DSSE achieves approximately 3-4x execution speed than existing schemes. In the future, the optimal block size will be investigated and the scenarios with different security levels will be updated.

Acknowledgements This work was supported by the Natural Science Foundation of Chongqing (Grant.cstc2018jcyjAX0510). The authors thank TopEdit (www.topeditsci.com) for its linguistic assistance during the preparation of this manuscr ipt.

Compliance with ethical standards

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

A Appendix

Here, the $Real^{\Sigma_{OSM}}_{\mathrm{A}}$ and $Ideal^{\Sigma_{OSM}}_{\mathrm{A},L,S}$ games for the semantic security described in Theorem 1 are introduced.

$Game_0$ (Algorithm 3): it is the same as $Real^{\Sigma_{OSM}}_{\mathrm{A}}$. At the beginning of the game, the adversary selects two sets of plaintext files of the same length and sends them to the challenger ($Game_0$: line 1). The challenger decides the encrypted file set by tossing a coin ($Game_0$: lines 2-3). The adversary outputs the keyword and file id ($Game_0$: lines 4-5) for the next search or performs update operations based on previous learning results. If op=search, the adversary generates a search token for searching. Then, the server reshuffles the path and change the next access path ($Game_0$: lines 6-9). If op=update, the adversary generates an updated token

Algorithm 3: $Game_0$: $b \leftarrow Real_{\mathrm{A}}^{\Sigma_{OSM}}(\lambda)$

1 $(st_{\mathrm{A}}, D_0, D_1) \leftarrow \mathrm{A}\left(1^{\lambda}\right)$
2 $b \xleftarrow{\$} \{0,1\}$
3 $(K, \delta_0, I', c_b) \leftarrow Setup\left(1^{\lambda}, D_b\right)$ // Generate key, construct an encryption incidence matrix and ciphertext collection
4 **for** $k = 1$ *to* q **do**
5 $\left(st_{\mathrm{A}}, op_k, \left(id_{0,k}, id_{1,k}\right), (w_{0,k}, w_{1,k})\right) \leftarrow \mathrm{A}\left(1^{\lambda}, st_{\mathrm{A}}, I', c_b, \tau_{b,1}, ..., \tau_{b,k-1}\right)$
6 **if** $op_k == search$ **then**
7 $\tau_{b,k} \leftarrow SearchToken\left(w_{b,k}\right)$ // Generate a search token
8 $(\delta_k, Resp(w_{b,k})) \xleftarrow{OT_n^1} Search(\tau_{b,k}, \delta_{k-1}; I', c_b)$ // Keyword search
9 $(I_k{}') \leftarrow Search.pathShuffling\left(I_{k-1}{}'\right)$ // Path shuffling
10 **else**
11 $op_k == update$
12 $\tau_{f_{b,k}} \leftarrow UpdateToken(K, id_{b,k})$ // Generate an update token
13 $(\delta_k; I_k{}', c_{b,k}) \leftarrow Update\left(\tau_{f_{b,k}}, \delta_{k-1}; I_{k-1}{}', c_{b,k-1}\right)$
14 **end**
15 **end**
16 let $\tau_b = \left(\tau_{b,1}, ..., \tau_{b,q}\right)$ //The $\tau_{b,k}$ represents the credential sent by the client to the server
17 $b^* \leftarrow \mathrm{A}(st_{\mathrm{A}}, \delta_k, I', c_b, \tau_b)$
18 **if** $b^* = b$ **then**
19 output 1
20 **else**
21 output 0
22 **end**

Algorithm 4: $\underline{Game_1}$: $\underline{\underline{Game_2}}$:

1 $(st_{\mathrm{A}}, D_0, D_1) \leftarrow \mathrm{A}\left(1^{\lambda}\right)$
2 $b \xleftarrow{\$} \{0,1\}$
3 $\underline{K \leftarrow \{k_1, k_2, k_3\} \xleftarrow{\$} \{0,1\}^{\kappa}}$ // Random value generates key
4 $(\delta_0, I', c_b) \leftarrow Setup(D_b)$ //construct an encryption incidence matrix and ciphertext collection
5 **for** $k = 1$ *to* q **do**
6 $\left(st_{\mathrm{A}}, op_k, \left(id_{0,k}, id_{1,k}\right), (w_{0,k}, w_{1,k})\right) \leftarrow \mathrm{A}\left(1^{\lambda}, st_{\mathrm{A}}, I', c_b, \tau_{b,1}, ..., \tau_{b,k-1}\right)$
7 **if** $op_k == search$ **then**
8 $\underline{\underline{\tau'_{b,k} \leftarrow SearchToken(Random(w_{b,k}))}}$ // Randomly select a keyword to generate a search token
9 $(\delta_k, Resp(w_{b,k})) \xleftarrow{\boxed{OT_n^1}} \mathrm{Search}\left(\underline{\underline{\tau'_{b,k}}}, \delta_{k-1}; I', c_b\right)$
10 $(I_k{}') \leftarrow Search.pathShuffling\left(I_{k-1}{}'\right)$ // Path shuffling
11 **else**
12 $op_k == update$
13 $\underline{\underline{\tau'_{f_{b,k}} \leftarrow UpdateToken(K, Random(\cdot))}}$ // Generate a column matrix with random values
14 $(\delta_k; I_k{}', c_{b,k}) \leftarrow Update\left(\tau_{f_{b,k}}, \delta_{k-1}; I_{k-1}{}', c_{b,k-1}\right)$
15 **end**
16 **end**
17 let $\tau_b = \left(\tau_{b,1}, ..., \tau_{b,q}\right)$
18 $b^* \leftarrow \mathrm{A}(st_{\mathrm{A}}, \delta_k, I', c_b, \tau_b)$
19 **if** $b^* = b$ **then**
20 output 1
21 **else**
22 output 0
23 **end**

to update the server ($Game_0$: lines 11-13). At the end of the game, adversary A outputs a bit b^* ($Game_0$: lines 16-17) based on previously learned transaction credentials and other information. When $b = b^*$, the game attacks success-

Algorithm 5: $\boxed{Game_4:}$ $b \leftarrow Ideal_{\mathrm{A},S,L}^{\Sigma_{OSM}}(\lambda)$

1 $(st_{\mathrm{A}}, D_0, D_1) \leftarrow \mathrm{A}\left(1^{\lambda}\right)$
2 $b \xleftarrow{\$} \{0,1\}$
3 $\boxed{(st_S, (I_M{}', I_T), c_b) \leftarrow SimStp\left(1^{\lambda}, L_{stp}, D_b\right)}$ // Generate encrypted index and ciphertext based on leakage function
4 $(\delta_0, I', c_b) \leftarrow Setup(D_b)$
5 **for** $k = 1$ *to* q **do**
6 $\left(st_{\mathrm{A}}, op_k, \left(id_{0,k}, id_{1,k}\right), (w_{0,k}, w_{1,k})\right) \leftarrow \mathrm{A}\left(1^{\lambda}, st_{\mathrm{A}}, I', c_b, \tau_{b,1}, ..., \tau_{b,k-1}\right)$
7 **if** $op_k == search$ **then**
8 $\boxed{\left(\tau_{b,k}^{s},\right) \leftarrow SearchToken\left(w_{b,k} \xleftarrow{\$} L_{srch}(m_l, N)\right)}$ // Generate a search token based on leakage function to search
9 $\boxed{(st_S; Resp^{s}(w_{b,k})) \leftarrow SimSrch\left(st_S; \tau_{b,k}^{s}; I', c_b\right)}$
10 $(I_k{}') \leftarrow Search.pathShuffling\left(I_{k-1}{}'\right)$
11 **else**
12 $op_k == update$
13 $\boxed{\tau_{f_{b,k}}^{s} \leftarrow UpdateToken\left(id_{b,k} \xleftarrow{\$} L_{upd}(ID_i, |c_i|)\right)}$ // Generate update token based on leakage function to update
14 $\boxed{(st_S; I_k{}', c_{b,k}) \leftarrow SimUpd\left(st_S; \tau_{f_{b,k}}^{s}; I_{k-1}{}', c_{b,k-1}\right)}$
15 **end**
16 **end**
17 let $\tau_b = \left(\tau_{b,1}, ..., \tau_{b,q}\right)$
18 $b^* \leftarrow \mathrm{A}(st_{\mathrm{A}}, \delta_k, I', c_b, \tau_b)$
19 **if** $b^* = b$ **then**
20 output 1
21 **else**
22 output 0
23 **end**

fully, $Game_0$ outputs 1; otherwise, it outputs 0 ($Game_0$: lines 18-21). Because $Game_0$ corresponds to a real experiment, $\Pr\left[Real_{\mathrm{A}}^{\Sigma_{OSM}}(\lambda) = 1\right] = \Pr\left[Game_0 = 1\right]$.

$Game_1$ (Algorithm 4): it is the same as $Game_0$, except that the value of the $K \leftarrow OSM\text{-}DSSE.Gen(1^{\kappa})$ is replaced with a randomly uniformly selected value in the setup phase ($\underline{Game_1}$: line 3). Because the random value is safe and indistinguishable from the key generated by the key generation algorithm, the search and update operations can still proceed normally. So, $\Pr\left[Game_1 = 1\right] - \Pr\left[Game_0 = 1\right] \leq negl(\lambda)$.

$Game_2$ (Algorithm 4): it is the same as $Game_1$, except that the determination function for generating the search or update token is replaced by a random function ($\underline{\underline{Game_2}}$: lines 8, 13). The random function is truly random and secure, and the output value is indistinguishable from the output value of the hash function (pseudo-random function). So, $\Pr\left[Game_2 = 1\right] - \Pr\left[Game_1 = 1\right] \leq negl(\lambda)$.

Moreover, since the 1-out-of-n OT protocol performed in search is based on the difficult problem of DDH ($\underline{\underline{Game_2}}$: lines 9), the client's choice is unconditionally secure. For any σ', there is r' that satisfies $y = g^{r'}h^{\sigma'}$. The client hides its choice in the token sent to the server by introducing a random number. So, the server cannot obtain any information about the client's choice from the token.

$Game_3$: it is the same as $Game_2$, except that the values used for homomorphic calculation in the shuffle phase are replaced with other randomly selected values for calculation. In this scheme, from the perspective of the server, the path shuffling algorithm invokes two parts: the confusion matrix M' of homomorphic encryption and the incidence matrix I' of symmetric encryption. The confusion matrix is composed of a permutation matrix P and a diagonal matrix Q that are randomly selected by the client. So, P and Q are not visible to the server. To declare the security of this part, the following theorem is given.

Theorem 2 *Even if the encrypted confusion matrix M' is given, the server cannot infer the permutation matrix P and the diagonal matrix Q.*

Proof: The security of the confusion matrix M' is based on the semantic security of Paillier encryption. The confusion matrix M does not reveal any information about P and Q. There are multiple choices of P and Q to generate the same confusion matrix M. These choices are not visible to the server, so the server cannot recognize the correct P and Q. The randomness of the matrix selection ensures that the server cannot correctly infer the true values of the P and Q, so the uploaded confusion matrix is safe.

For the server-side incidence matrix I', symmetric encryption is performed to meet the IND-CPA security standards.

According to theorem 2 and the above analysis on security, both parties involved in a homomorphic calculation are secure. At the same time, according to the homomorphic properties of the Paillier encryption system (i.e., any

calculation performed by the homomorphic operation can protect the privacy of the original data and the calculation result), the calculation result is also secure, and the server cannot correctly distinguish the real confusion matrix from the randomly generated confusion matrix. So, the equation $\Pr[Game_3 = 1] - \Pr[Game_2 = 1] \leq negl(\lambda)$ is obtained.

$Game_4$ (Algorithm 5): it is the same as $Game_3$, except that the output of the setup, search and update phases are replaced by the output of the simulator $SimStp(\cdot)$, $SimSrch(\cdot)$ and $SimUpd(\cdot)$ ($\boxed{Game_4}$: lines 3, 9, 14). According to the above analysis, the output of the simulator and $Game_3$ is indistinguishable, so $\Pr[Game_4 = 1] - \Pr[Game_3 = 1] \leq negl(\lambda)$ can be obtained. Because $Game_4$ is a game under the ideal experiment, $Game_3$ is indistinguishable from the ideal experiment.

Through the above games that include $Game_0$, $Game_1$, $Game_2$, $Game_3$, and $Game_4$, it can be obtained that

$$\left|\Pr\left[Real_{\mathrm{A}}^{\Sigma_{OSM}}(\lambda) = 1\right] - \Pr\left[Ideal_{\mathrm{A},S,L}^{\Sigma_{OSM}}(\lambda) = 1\right]\right| \leq negl(\lambda)$$

Therefore, the scheme proposed in this paper provides adaptive semantic security.

References

1. Song, D.X.,Wagner, D.,Perrig, A. Practical techniques for searches on encrypted data. In *Proc. - IEEE Symp. Secur. Privacy. SP 2000. S&P 2000*, pages 44–55. IEEE, 2000.
2. Goh, E.J. et al. Secure indexes. *IACR Cryptol. ePrint Arch.*, 2003:216, 2003.
3. Curtmola, R.,Garay, J.,Kamara, S.Ostrovsky, R. Searchable symmetric encryption: Improved definitions and efficient constructions. 2006.
4. Kamara, S.,Papamanthou, C.,Roeder, T. Dynamic searchable symmetric encryption. In *In Proc ACM Conf Computer Commun Secur*, pages 965–976, 2012.
5. Islam, M.S.,Kuzu, M.,Kantarcioglu, M. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *Ndss*, volume 20, page 12. Citeseer, 2012.
6. Cash, D., Grubbs, P., Perry, J., Ristenpart, T. Leakage-abuse attacks against searchable encryption. In *In Proc ACM Conf Computer Commun Secur*, pages 668–679, 2015.
7. Liu, C., Zhu, L., Wang, M., Tan, Y.A. Search pattern leakage in searchable encryption: Attacks and new construction. *Inf Sci*, 265:176–188, 2014.
8. Zhang, Y., Katz, J.,Papamanthou, C. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *In Proc. USENIX Secur. Symp. (USENIX Security 16)*, pages 707–720, 2016.
9. Oya, S., Kerschbaum, F. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. *arXiv preprint arXiv:2010.03465*, 2020.
10. Stefanov, E., Papamanthou, C., Shi, E. Practical dynamic searchable encryption with small leakage. In *NDSS*, volume 71, pages 72–75, 2014.
11. Bost, R., Minaud, B., Ohrimenko, O. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proc ACM Conf Computer Commun Secur. ACM*, pages 1465–1482, 2017.
12. Sun, S.F., Yuan, X., Liu, J.K., Steinfeld, R., Sakzad, A., Vo, V., Nepal, S. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *Proc ACM Conf Computer Commun Secur*, pages 763–780, 2018.
13. Goldreich, O., Ostrovsky, R. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.
14. Zhang, J., Ma, Q., Zhang, W., Qiao, D. Tskt-oram: A two-server k-ary tree oram for access pattern protection in cloud storage. In *In: Proc IEEE Mil Commun Conf MILCOM. IEEE*, pages 527–532. IEEE, 2016.
15. Demertzis, I., Papadopoulos, D.,Papamanthou, C., Shintre, S. {SEAL}: Attack mitigation for encrypted databases via adjustable leakage. In *Proc. USENIX Secur. Symp.*, 2020.
16. Garg, S., Mohassel, P., Papamanthou, C. Tworam: Round-optimal oblivious ram with applications to searchable encryption. *IACR Cryptol. ePrint Arch.*, 2015:1010, 2015.
17. Kamara, S., Moataz, T., Ohrimenko, O. Structured encryption and leakage suppression. In *In Lect. Notes Comput. Sci*, pages 339–370. Springer, 2018.
18. Chen, G., Lai, T.H., Reiter, M.K., Zhang, Y. Differentially private access patterns for searchable symmetric encryption. In *IEEEProc IEEE INFOCOM* , pages 810–818. IEEE, 2018.
19. Patel, S., Persiano, G., Yeo, K., Yung, M. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proc ACM Conf Computer Commun Secur*, pages 79–93, 2019.
20. Hoang, T., Yavuz, A.A., Guajardo, J. Practical and secure dynamic searchable encryption via oblivious access on distributed data structure. In *ACM Int. Conf. Proc. Ser. ACM*, pages 302–313, 2016.
21. Hoang, T., Yavuz, A.A., Durak, F.B., Guajardo, J. Oblivious dynamic searchable encryption on distributed cloud systems. In *Lect. Notes Comput. Sci. Springer*, pages 113–130. Springer, 2018.
22. Akavia, A., Gentry, C., Halevi, S., Leibovich, M. Setup-free secure search on encrypted data: Faster and post-processing free. *PoPETs*, 2019(3):87–107, 2019.
23. Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *Lect. Notes Comput. Sci.*, pages 223–238. Springer, 1999.
24. Zhang, Z., Wang, K., Lin, W., Fu, A.W.C., Wong, R.C.W. Repeatable oblivious shuffling of large outsourced data blocks. In *SoCC - Proc. ACM Symp. Cloud Comput.*, pages 287–298, 2019.
25. Wang, C., Cao, N., Li , J., Ren, K., Lou, W. Secure ranked keyword search over encrypted cloud data. In *In Proc Int Conf Distrib Comput Syst*, pages 253–262. IEEE, 2010.
26. Cao, N., Wang, C., Li, M., Ren, K., Lou, W. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *IEEE Trans Parallel Distrib Syst*, volume 25, pages 222–233. 2013.
27. Sun, W., Wang, B., Cao, N., Li, M., Lou, W., Hou, Y.T., Li, H. Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In *ASIA CCS - Proc. ACM SIGSAC Symp. Inf., Comput. Commun. Secur.*, pages 71–82. 2013.
28. Zhou, R., Zhang, X., Du, X., Wang, X., Yang, G., Guizani, M. File-centric multi-key aggregate keyword searchable encryption for industrial internet of things. In *IEEE Trans. Ind. Inf.*, volume 14, pages 3648–3658. 2018.
29. Cui, H., Deng, R.H., Lai, J., Yi, X., Nepal, S. An efficient and expressive ciphertext-policy attribute-based encryption scheme with partially hidden access structures, revisited. In *ProvSec.*, volume 133, pages 157–165. 2018a.

30. Xia, Z., Wang, X., Sun, X., Wang, Q. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. In *IEEE Trans Parallel Distrib Syst*, volume 27, pages 340–352. 2016.
31. Cui, H., Wan, Z., Deng, R., Wang, G., Li, Y. Efficient and expressive keyword search over encrypted data in the cloud. In *IEEE Trans. Dependable Secure Comput.*, volume 15, pages 409–422. 2018b.
32. Asharov, G., Naor, M., Segev, G., Shahaf, I. Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations. In *STOC 2016*, pages 1101–1114. 2016.
33. Demertzis, I., Papamanthou, C. Fast searchable encryption with tunable locality. In *SIGMOD.*, pages 1053–1067. 2017.
34. Bost, R. $\sum \varphi o \varsigma$: Forward secure searchable encryption. In *Proc ACM Conf Computer Commun Secur. ACM*, pages 1143–1154. 2016.
35. Zhang, R., Xue, R., Yu, T., Liu, L. Pvsae: A public verifiable searchable encryption service framework for outsourced encrypted data. In *Proc. - IEEE Int. Conf. Web Serv., ICWS. IEEE*, pages 428–435. IEEE, 2016b.
36. Zhang, J., Song, C., Wang, Z., Yang, T., Ma, W. Efficient and provable security searchable asymmetric encryption in the cloud. In *IEEE Access*, volume 6, pages 68384–68393. 2018.
37. Liu, C., Zhu, L., Chen, J. Graph encryption for top-k nearest keyword search queries on cloud. In *IEETrans. Sust. Comp.*, volume 2, pages 371–381. 2017.
38. Kamara, S., Moataz, T. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Lect. Notes Comput. Sci.*, pages 94–124. Springer, 2017.
39. Ge, X., Yu, J., Hu, C., Zhang, H., Hao, R. Enabling efficient verifiable fuzzy keyword search over encrypted data in cloud computing. In *IEEE Access*, volume 6, pages 45725–45739. 2018.
40. Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., Steiner, M. Dynamic searchable encryption in very-large databases: data structures and implementation. In *NDSS*, volume 14, pages 23–26. Citeseer, 2014.
41. Naveed, M. The fallacy of composition of oblivious ram and searchable encryption. *IACR Cryptol. ePrint Arch.*, 2015:668, 2015.
42. Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310, 2013.
43. Tzeng, W.G. Efficient 1-out-of-n oblivious transfer schemes with universally usable parameters. *IEEE Transactions on Computers*, 53(2):232–240, 2004.
44. Warner, S.L. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.
45. Cohen, W. The cmu enron email dataset. *URL http://www. cs. cmu. edu/˜ enron/, Accessed*, 12, 2006.
46. Porter, M.F., et al. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
47. Hoang, T., Yavuz, A.A., Guajardo, J. A high-security searchable encryption framework for privacy-critical cloud storage services. *IACR Cryptol. ePrint Arch.*, 2017:1237, 2017.

Figures

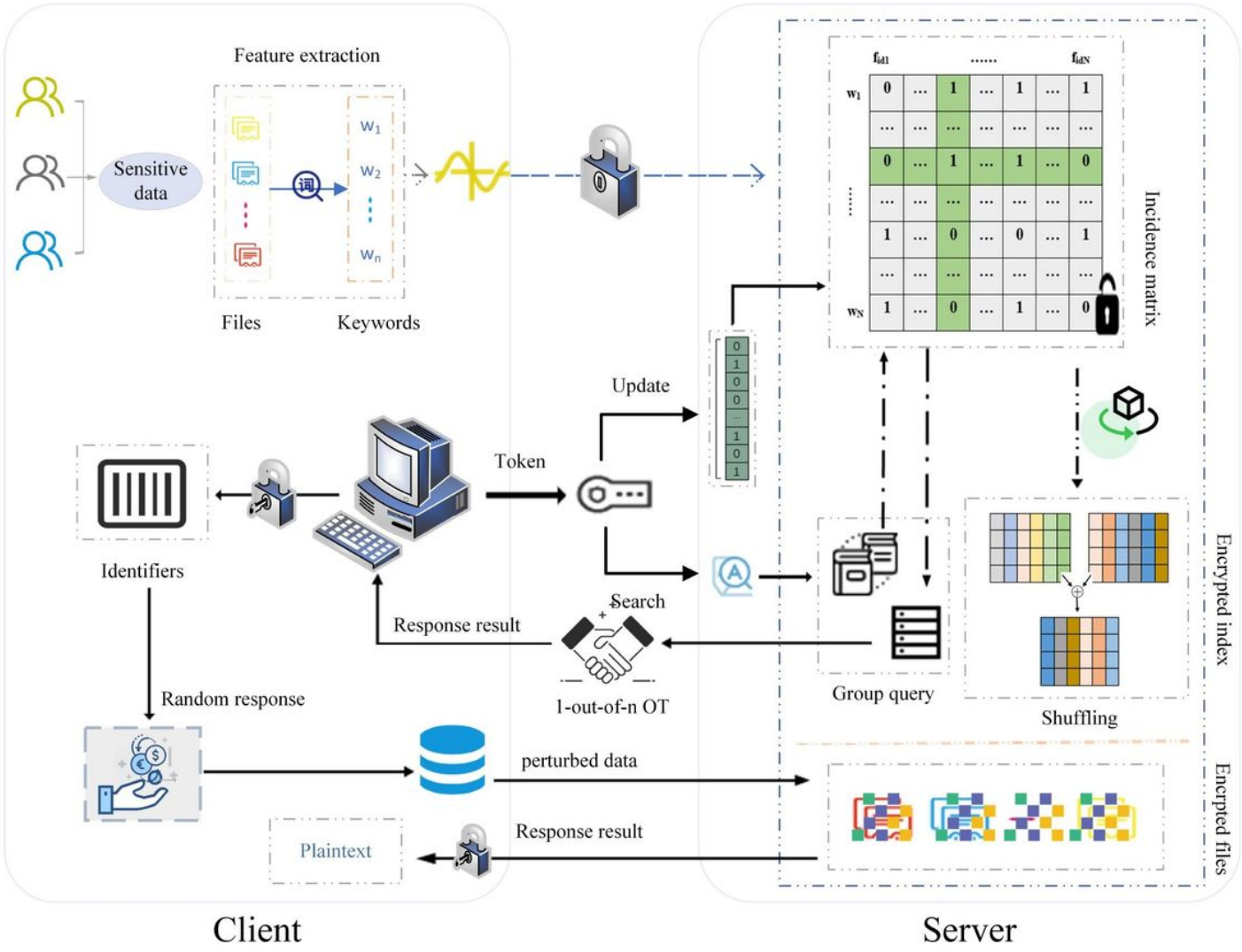


Figure 1

The OSM-DSSE system model

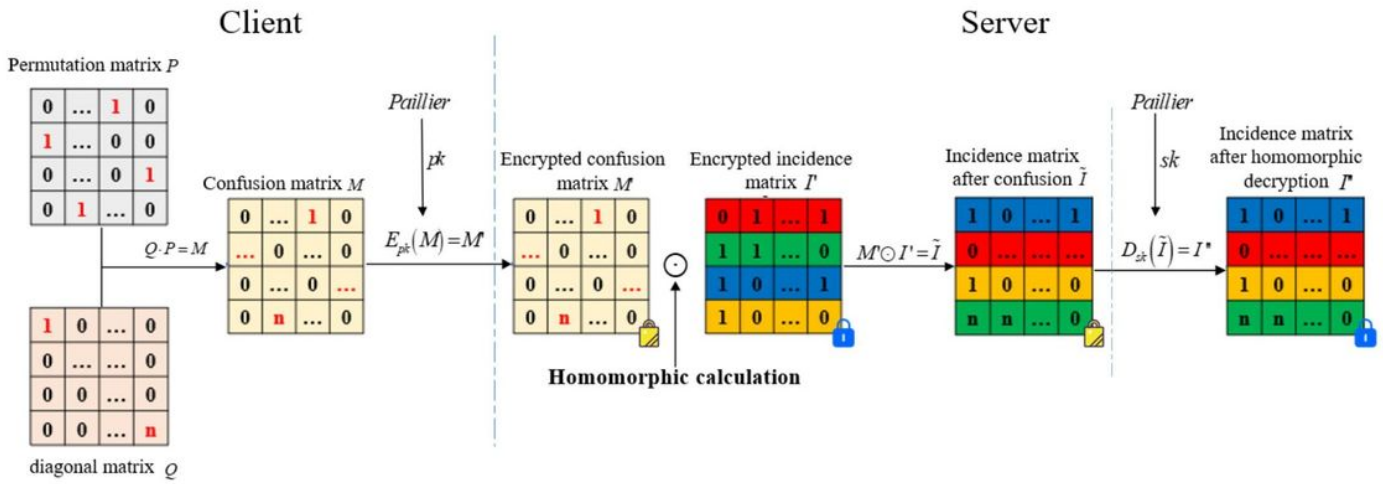


Figure 2

Shuffling process

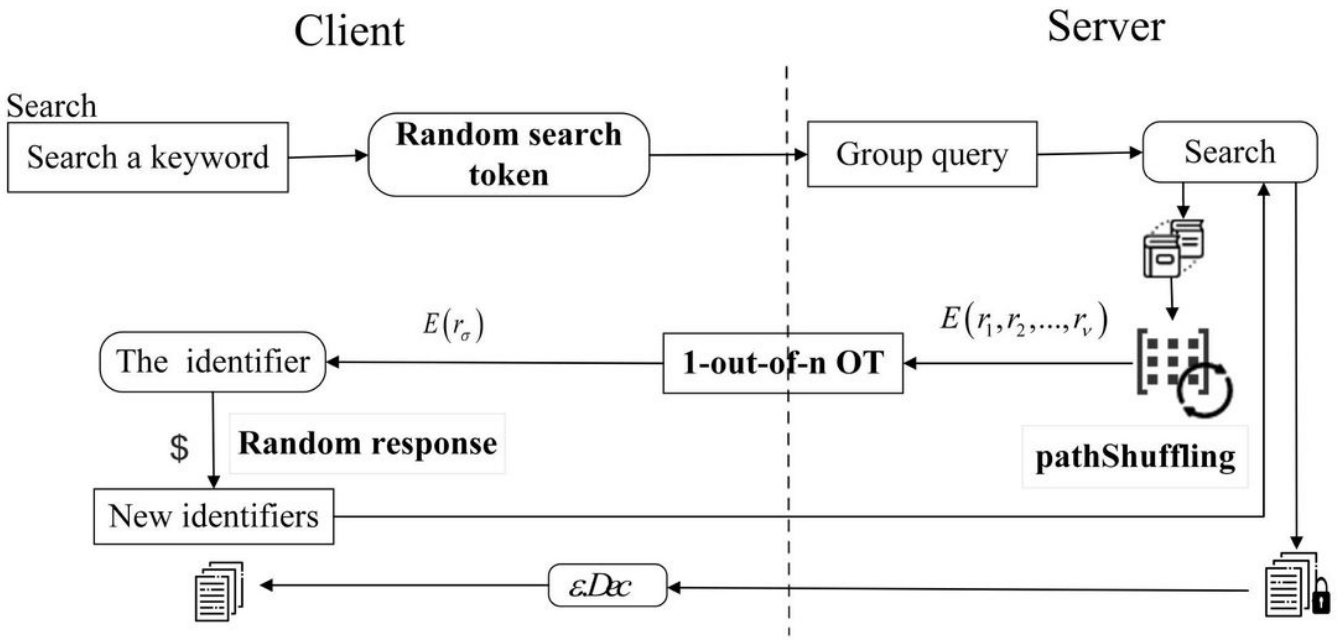


Figure 3

Hidden search pattern diagram

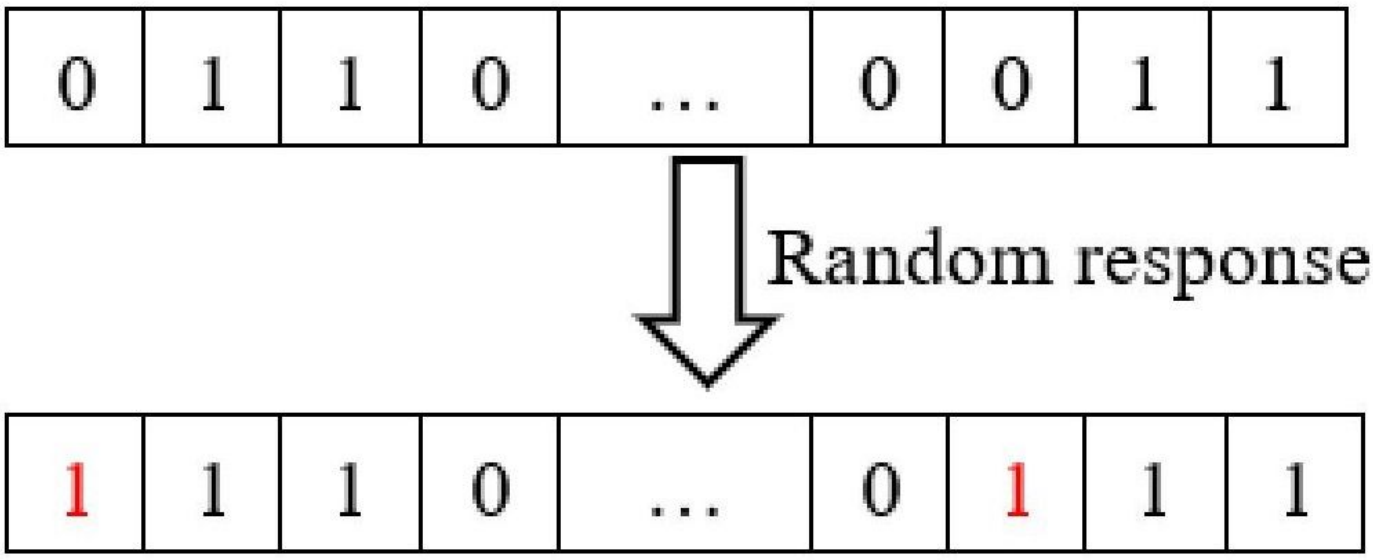


Figure 4

Random response

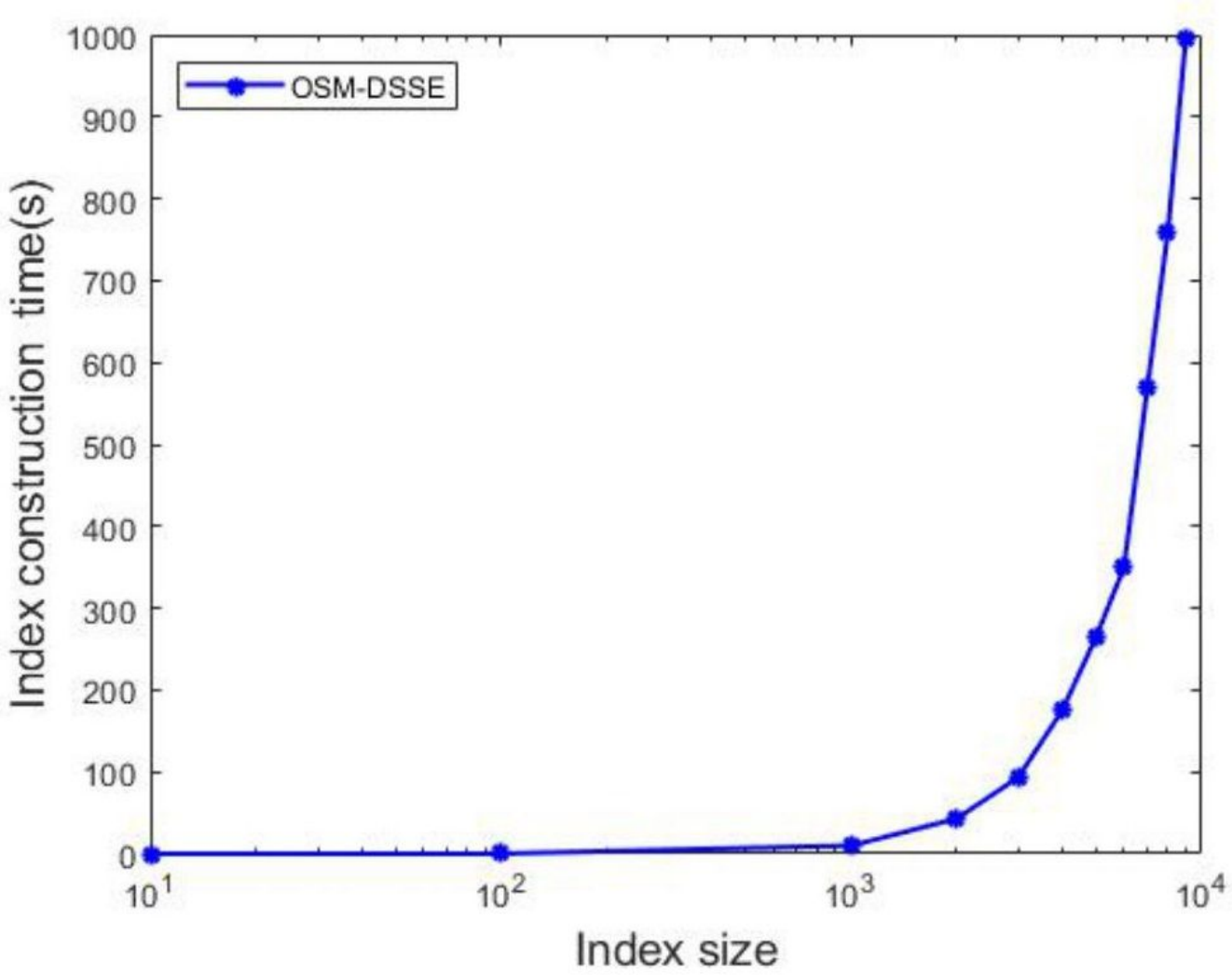


Figure 5

The relation between index size and construction time

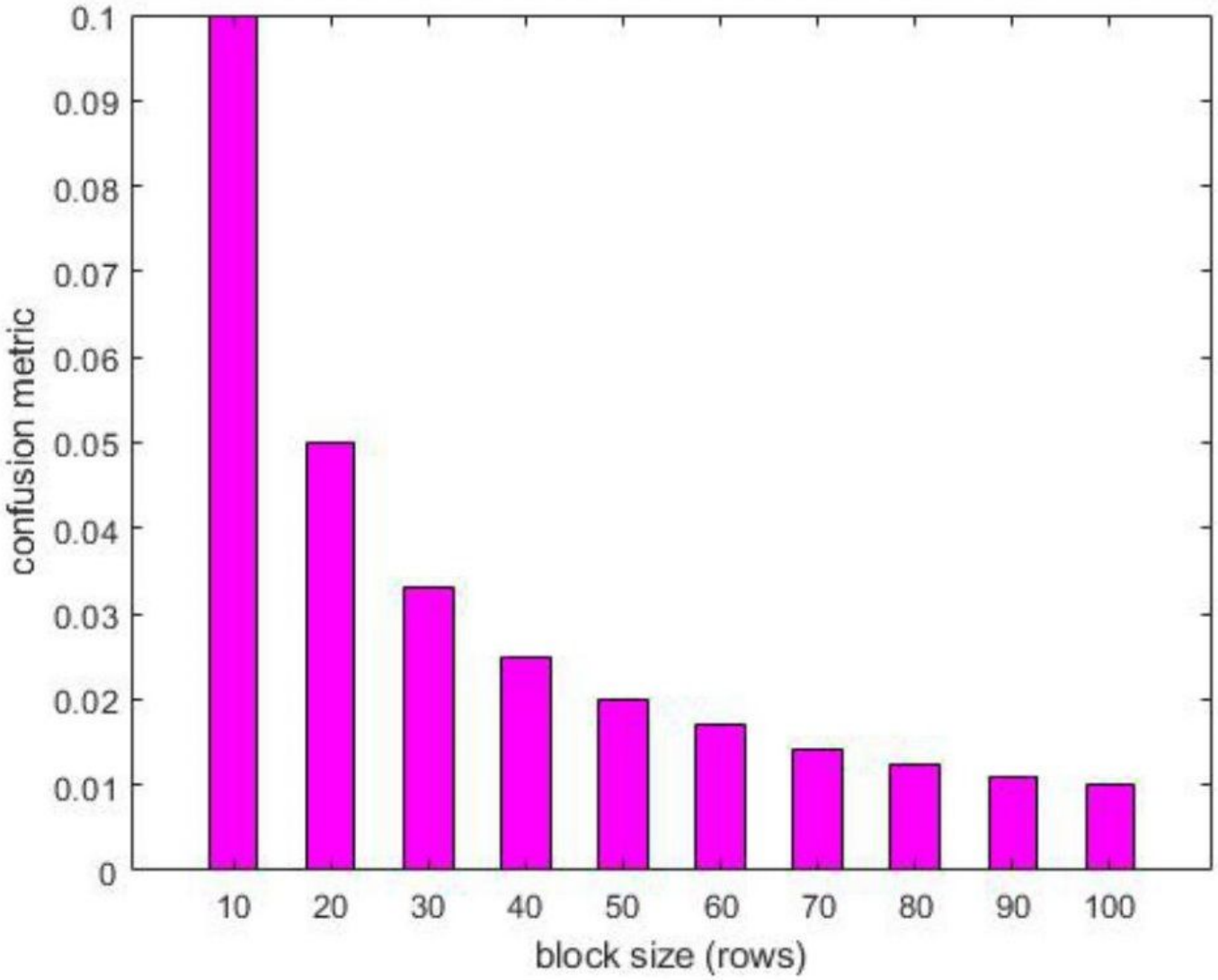


Figure 6

The relation between confusion metric and block size

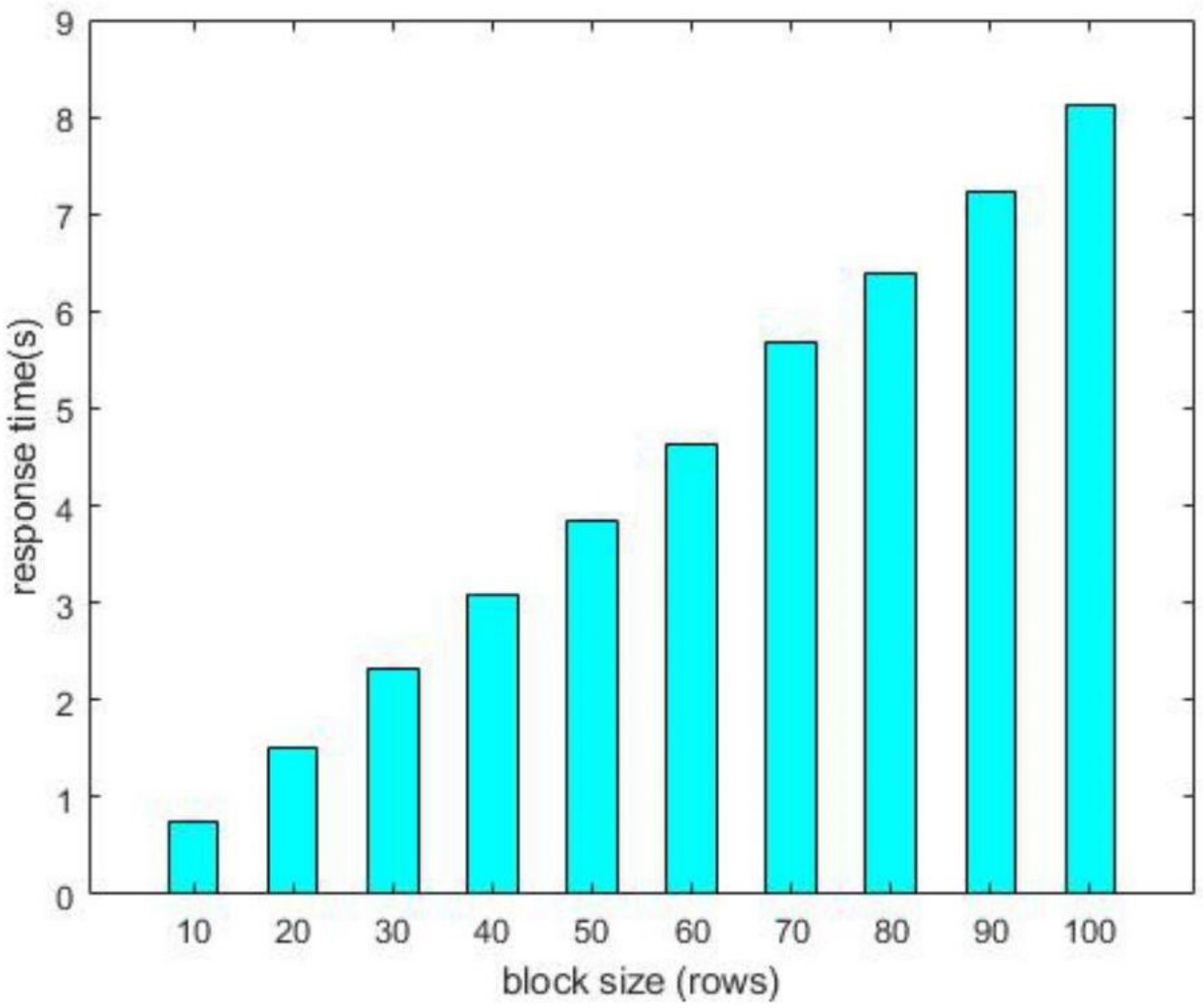


Figure 7

The relation between response time and block size

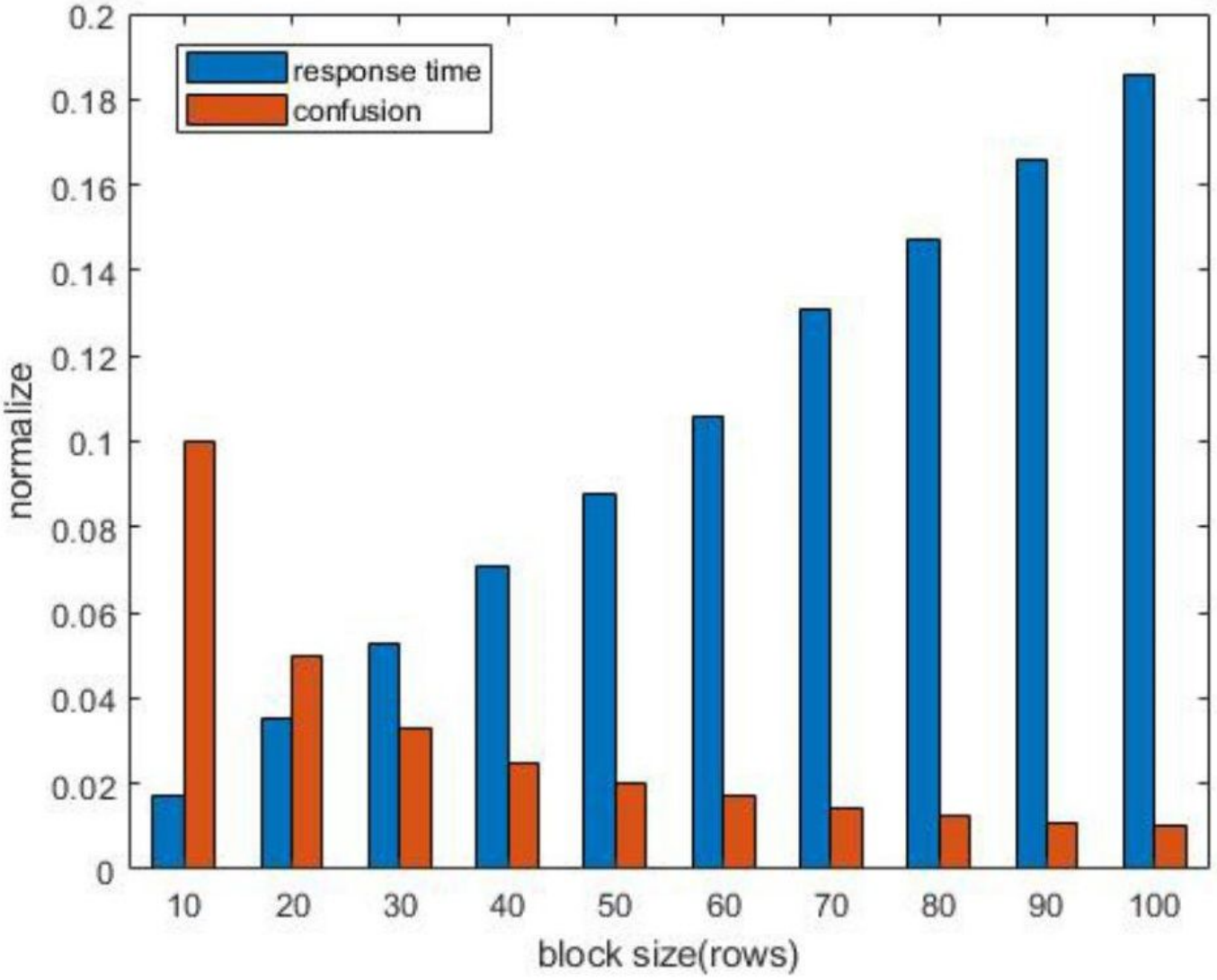
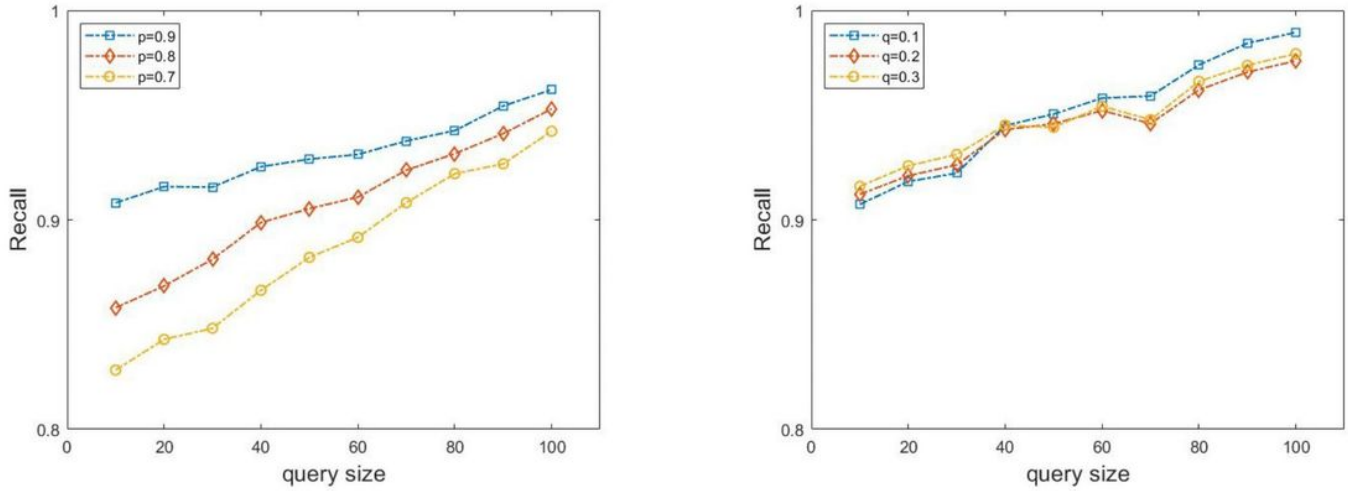


Figure 8

The comparison between confusion and response time



(a) Fix q=0.1 (b) Fix p=0.9

Figure 9

The effect of returned result size on recall

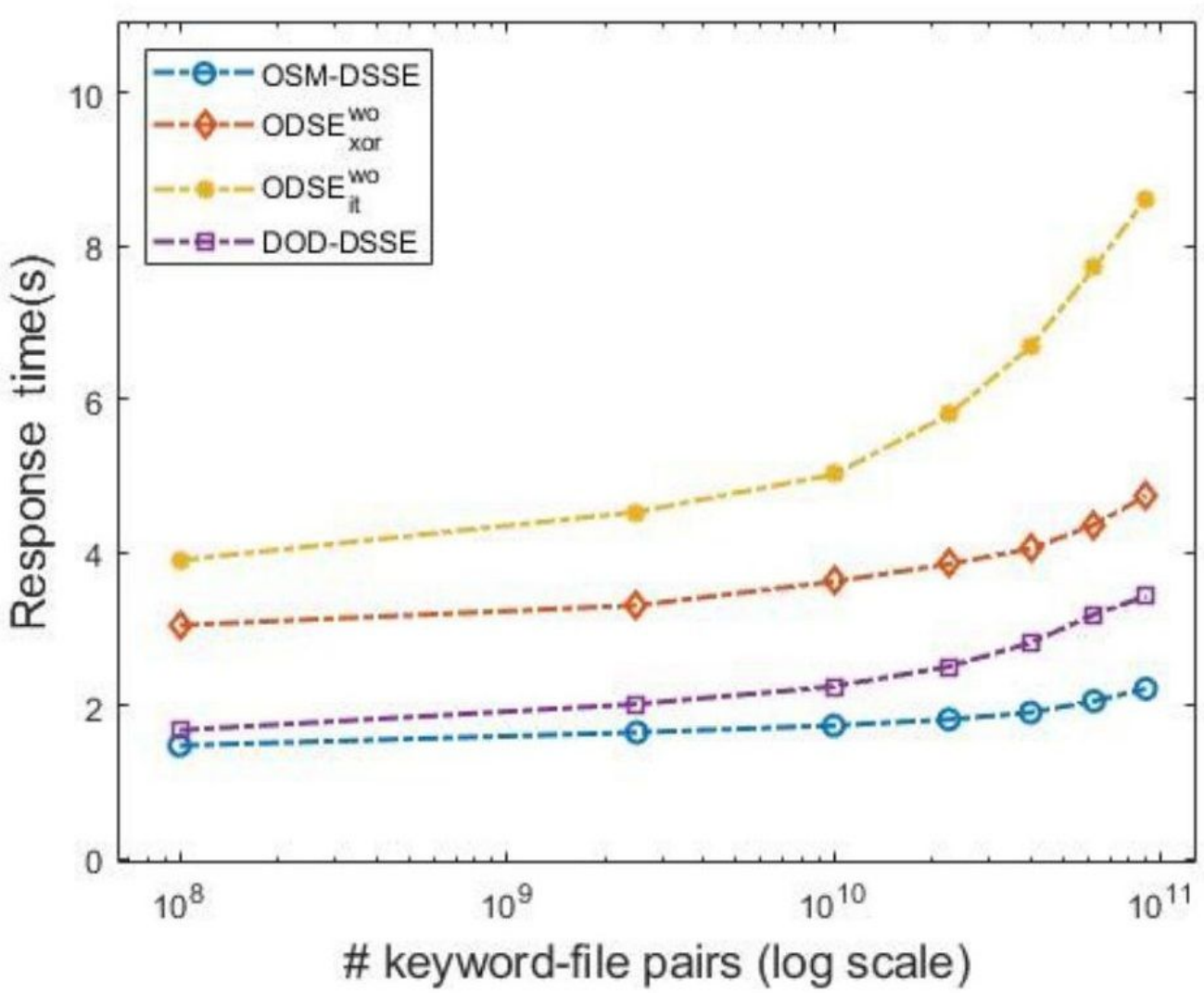


Figure 10

The relation between search time and number of keyword-file pairs

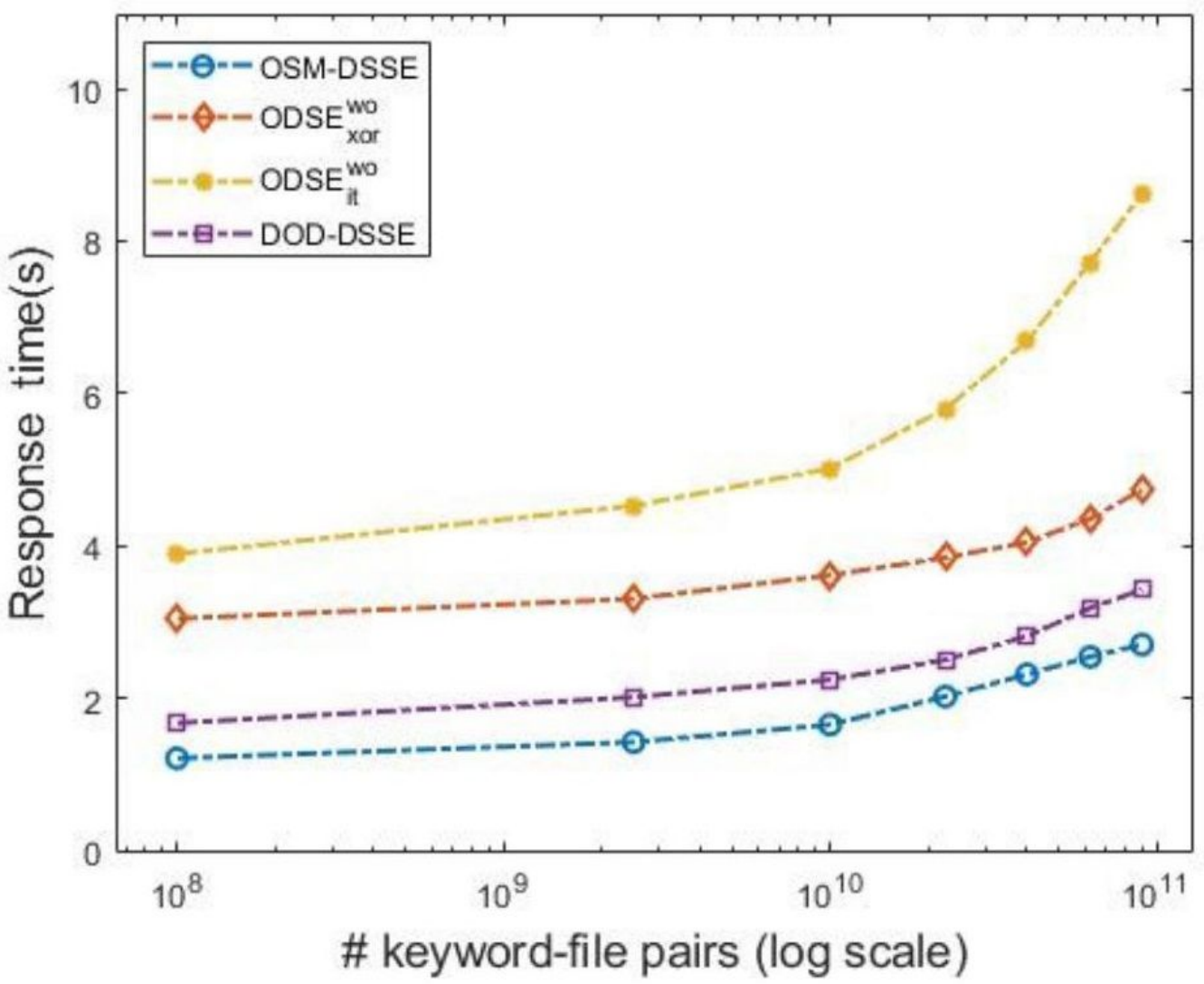


Figure 11

The relation between update time and number of keyword-file pairs