A Searchable Encryption Scheme with Hidden Search Pattern and Access Pattern on Distributed Cloud System

Shengchen Duan
Beijing University of Posts and Telecomunications
Hong Liu (liliary@163.com)
Chongqing University of Posts and Telecommunications
Lize Gu
Beijing University of Posts and Telecomunications

Research Article

Keywords: Dynamic Searchable Encryption, Search Pattern, Access Pattern, Paillier Encryption, Optimised Padding

Posted Date: October 18th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2158129/v1

A Searchable Encryption Scheme with Hidden Search Pattern and Access Pattern on Distributed Cloud System

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. 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. 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 without using ORAM. The OSM-DSSE scheme realizes efficient search and update operations based on an incidence matrix. In particular, a shuffling algorithm using Paillier encryption combines the 1-out-of-n obliviously transfer (OT) protocol to hide access pattern and an optimized padding scheme to obfuscate the search pattern with small storage overhead. Besides, a formalized security analysis and performance analysis of 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.

Keywords: Dynamic Searchable Encryption, Search Pattern, Access Pattern, Paillier Encryption, Optimised Padding

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, a malicious user may be tampered with or stolen sensitive information. 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. Furthermore, the 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 new encryption primitive, searchable encryption enables the user to search for a keyword over the ciphertext. However, the application was limited to searching on static encrypted data and could not resist the simple adversary attack. In 2003, Goh et al. [2] formally defined the secure incidence and developed a security model called the "semantic security" for adaptive selective keyword attacks. However, the accuracy of the 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 security" 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 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 user data and query information. For instance, Islam et al. [5] and Cash et al. [7] firstly exploited access pattern leakage and prior knowledge about the dataset to recover the user's query information. Liu et al. [6] 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. [44] 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.

Although some solutions have been proposed for the leakages and attacks described above, most of the research focuses on forward-secure and backward-secure methods [9-11]. The ORAM can address the problem of access pattern leakage [12,13,45], but it was impractical for widespread adoption. Garg et al. [14] exploited ORAM and garbled RAM (Random Access Memory) to hide the search pattern. Kamara et al. [15] 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. To overcome the shortcomings of ORAM, Wang et al. [16] proposed a scheme that can hide the search pattern without utilizing ORAM and has high efficiency. However, search operation needs to be done using the auxiliary server, leading to extra costs. Dauterman et al. [17] leverage linear scans to build a system that can achieve better performance for excepted workloads, but they do not implement features expected from a search engine such as finding non-exact matches, ranking results, or providing summaries. 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 must 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. [18,19] 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. [20] proposed a secure search using FHE on an encrypted data structure. 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 incidence (explicit) without considering the information leakage in accessing the file (implicit).

This paper proposes a new dynamic searchable encryption scheme called oblivious shuffle incidence matrix DSSE (OSM-DSSE) 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:

1. This paper proposes a shuffling algorithm with Paillier encryption [22,23] to address the problem of access pattern leakage, which can shuffle the data in the incidence matrix to change the access path.
2. 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.
3. Since searching the same keyword always returns the same file set, exposing the response length is disclosed. This paper utilizes an optimized padding scheme to hide the response length. Meanwhile, a database padding algorithm is proposed, which uses the combination of optimal padding and clustering algorithm to reduce the redundant storage of data and hide the search pattern completely with the minimum storage overhead.

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

2 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 the first scheme with optimal search complexity, where the number of documents containing the keyword w. Many improvements have been made in their subsequent work [24-26]. 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 [15,27,28], dynamics [9,29,30], localization [31,32], security [33-35], and complex functions [36-38], they still suffer from leaking some important information. Attackers can use these leakages to attack and recover data and cause more serious information leakages [5-8].

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 link ability of newly added data and query keywords; backward-secure means that the server can no longer match and retrieve the deleted data. Stefanov et al. [9] proposed the first solution to support forward-secure property, but it exhibits a linear time complexity for the search. Bost et al. [10] 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. [11] 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.

Oya et al. [21] point out that, beyond access pattern, SSE also leaks search pattern leakage which can further be leveraged to realize query recovery. However, the effectiveness of this attack relies on a strong assumption of fully knowing the query frequency in the real world. Besides, the accuracy of their attack based on maximum likelihood estimation is also probabilistic, and it significantly depends on the query distribution. Recent attacks do not require strong knowledge assumptions on plaintext data, but the volume pattern is essential for their accuracy.

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. [12] 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. [13] proposed a method based on ORAM access pattern protection in the cloud storage environment. Garg et al. [14] proposed a TWORAM scheme that reduces the client storage overhead while hiding the file access pattern with ORAM. Demertzis et al. [45] present SEAL, a framework for encrypted databases with improved security via a light use of ORAM and padding. It is open whether our attack applies in such modified settings. However, the researches have shown that using ORAM to eliminate information leaks leads to high overhead and low execution efficiency [39-41].

To overcome the shortcomings of ORAM, many researchers proposed schemes which can hide the search pattern with high efficiency. Wang et al. [16] hide the search pattern in order to guarantee trapdoor and keyword privacy. They used a particular type of additive homomorphic encryption to accomplish the conjunctive keyword searchable encryption. To meet the privacy aims of the strategy, they proposed two servers: a cloud server and an auxiliary server. They also used random polynomials to improve user security. Their approach ensures a higher level of protection for cloud users. However, search operation needs to be done using the auxiliary server, making this method not so efficient.

3 Overview of OSM-DSSE Scheme

3.1 System goal

We aim to effectively perform a privacy-protected keyword search and file update on an encrypted cloud database. 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
  1) Based on the group query, the server utilizes the two-level map to obtain the target data block containing multiple pieces of data. Furthermore, 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. It can also 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. Therefore, this paper utilizes an optimised padding strategy based on cluster algorithm to hide the response length (implicit search pattern).

3.2 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 inci dence 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. Even though data files are encrypted during the access, 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.

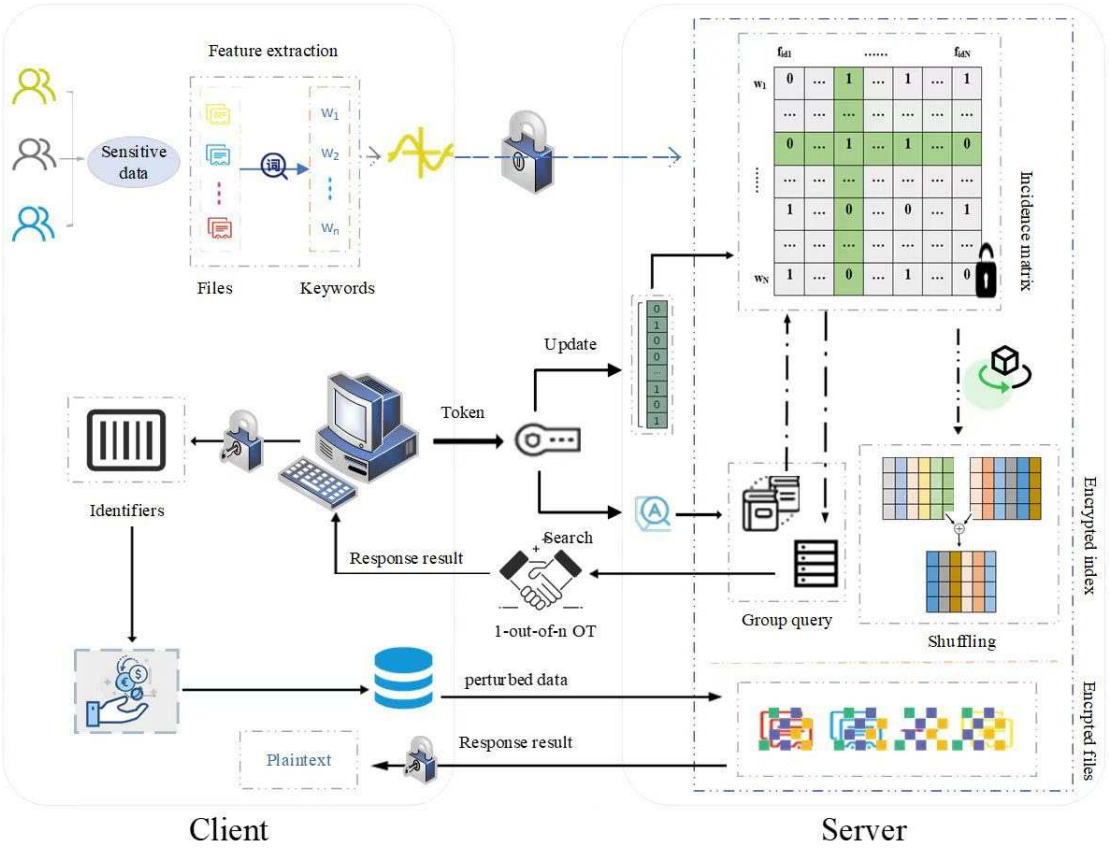

Fig 1 OSM-DSSE system model

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 the same access path easily identifies a repeated query. When the client sends a search request to the server, repeated searches also lead to

disclosing 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 reducing the computational and communicational cost of the searchable encryption scheme. The existing schemes [18,41] usually use the "fetch-decrypt-reencrypt-upload" strategy to hide the access pattern, since data volume of searchable encryption schemes is usually huge, it causes high communication and computation overhead. Although we can adaptively utilize "fetch-decrypt-reencrypt-upload" strategy, e.g., just reencrypt the access path for one search and several dummy access paths instead of the whole data structure, it leaks important information by statistical attack. Fortunately, the matrix data structure ensures us to 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, shown in Fig. 2.

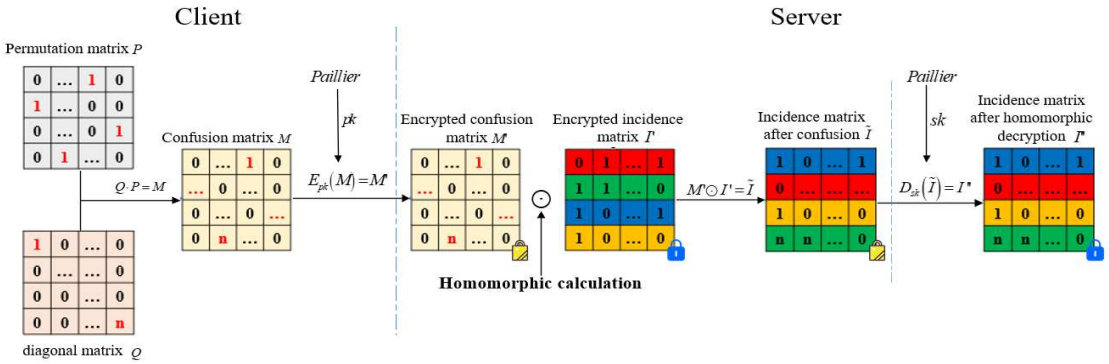

Fig. 2 Shuffling process. The yellow lock indicates homomorphic encryption, and the blue lock indicates the symmetric encryption

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 $\pi=\begin{pmatrix}01\\10\end{pmatrix}$:

$$\pi\cdot B=\begin{pmatrix}01\\10\end{pmatrix}\cdot\begin{pmatrix}B_1\\B_2\end{pmatrix}=\begin{pmatrix}B_2\\B_1\end{pmatrix} \quad (1)$$

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

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

3) Based on the formulas (1) (2), the confusion matrix can be obtained:

$$M=Q\cdot P \quad (3)$$

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

$$M'\leftarrow E_{pk}(M) \quad (4)$$

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.

$$\%\leftarrow M'\text{e } I' \quad (5)$$

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.

$$I'' \leftarrow D_{sk}(I') \tag{6}$$

3.3.2 Hiding the search pattern

Simon et al. [21] 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. It is challenging to hide the search pattern while significantly reducing the communicational cost of the searchable encryption scheme. Originally padding scheme can effective hides implicit search pattern, but it pads result set of every keyword to the maximum length which means it causes huge storage overhead. This paper proposed an optimized padding scheme by using a cluster algorithm and calculating the optimal padding length without leaking private information.

Therefore, both the explicit and the implicit search patterns are hidden in this paper.

Implicit search pattern hiding In this paper, we design an optimized padding scheme which significantly reduces server storage overhead to hide the implicit search pattern. We will introduce details of the optimized padding scheme as follow.

Padding the response length of each keyword to their maximum response length is a native approach to designing a volume-hiding encryption scheme. It is easy to see that this hides the response lengths. Unfortunately, it also induces a non-trivial storage overhead. So we propose an optimized padding method to overcome this shortcoming.

Assuming that the number of keywords in the database is N, and to prevent attacks based on response length, the database owner can pad the real responding volume distribution by inserting dummy files. More formally, when responding to a query for the keyword w, the CS will see that it matches $N_w^o = N_w^r + N_w^p$ files, where N_w^r is the number of real files matched by w, and N_w^p is the number of confused files used to pad. Similarly, the total number of files in the padded database is $N^o = N^r + N^p$, where N^r is the number of real files, and N^p is the number of disturbing files. Then the observed probability e_w^o, real probability e_w^r, and confusion probability e_w^p of keywords can be expressed as $e_w^o = E[\frac{N_w^o}{N^o}]$, $e_w^r = E[\frac{N_w^r}{N^r}]$, and $e_w^p = E[\frac{N_w^p}{N^p}]$. We denote γ representing the ratio of the inserted dummy file to the real file: $\gamma = \frac{N^r}{N^p}$. By combining this with $N_w^o = N_w^r + N_w^p$, we can obtain the relationship between e_w^o, e_w^r, and e_w^p:

$$e_w^o = \frac{1}{1+\gamma} e_w^r + \frac{\gamma}{1+\gamma} e_w^p \Leftrightarrow e_w^p = (1+\frac{1}{\gamma}) e_w^o - \frac{1}{\gamma} e_w^r \tag{3.1}$$

Since e_w^p must be between 0 and 1, the lower bound of γ can be obtained:

$$\gamma \geq \max_{w \in W} \{\frac{e_w^r - e_w^o}{e_w^o}, \frac{e_w^r - e_w^o}{e_w^o - 1}\}.$$

In order to achieve the optimal padding to minimize γ, we utilized the classical k-means clustering algorithm to cluster keywords according to their response length. For a given optimal cluster $\Gamma = (G_1, ..., G_m)$, when $w \in G_i$, $e_{\max(i)}^r$ represents the maximum value of e_w^r, and $e_{\min(i)}^r$ represents the minimum value of e_w^r. According to Equation (3.1), the minimum value of γ can be calculated:

$$\gamma_{\min}(\Gamma, \tilde{e}) = \max_{w \in W} \{\frac{e_w^r - e_w^o}{e_w^o}, \frac{e_w^r - e_w^o}{e_w^o - 1}\} = \max_{1 \leq i \leq m} \{\max_{w \in G_i} \{\frac{e_w^r - \tilde{e}_i}{\tilde{e}_i}, \frac{e_w^r - \tilde{e}_i}{\tilde{e}_i - 1}\}\}$$

$$= \max_{1 \leq i \leq m} \{\frac{e_{\max(i)}^r}{\tilde{e}_i}, \frac{1 - e_{\min(i)}^r}{1 - \tilde{e}_i}\} - 1 = (\sum_{i=1}^{m} |G_i| e_{\max(i)}^r) - 1 \tag{3.2}$$

Therefore, γ is the minimum when $\frac{e_{\max(i)}^r}{\tilde{e}_i} = \frac{1 - e_{\min(i)}^r}{1 - \tilde{e}_i}$: i.e., in order to achieve the optimal padding, the expected observed probability of each cluster should be $\tilde{e}_i = \frac{e_{\max(i)}^r}{1+\gamma}$. Note that the keyword will pad to the expected observed probability $\tilde{e}_i$ with the group to which it belongs. For example, the database

$I=\{I_1,...,I_{28}\}$ can be represented by seven result sets of keywords. According to the response length of keywords, they can be divided into two categories: {1,2,2} and {5,5,6,7}. Equation (3.2) gives us $\gamma_{\min}=\frac{3}{14}$, then $\tilde{e}_1=\frac{1}{17}$, $\tilde{e}_2=\frac{7}{34}$. The database $I=\{I_1,...,I_{28}\}$ will be padded with $I^o=\{I_1,...,I_{34}\}$, and the response length of the seven keywords will be padded with {2,2,2} and {7,7,7,7}.

An optimized padding scheme $OptPad=(ResNum,Cluster,PadNum,Pad)$ is a tuple that consists of six polynomial-time algorithms:

- $L\leftarrow ResNum(I)$ is a probabilistic algorithm to calculate the length of result set of each keyword. It takes a random incidence matrix I as input and output the L which L_i is the length of result set of a keyword w_i.
- $\Gamma\leftarrow Cluster(L)$ is a probabilistic algorithm to cluster keywords according to their response length, it takes L as input and output optimal cluster $\Gamma=(G_1,...,G_m)$.
- $pn\leftarrow PadNum(\Gamma)$ is a probabilistic algorithm to calculate the number of dummy files of each keyword w_i according to equation 3.2, it takes Γ as input and output number of dummy files of each keyword pn.
- $I'\leftarrow Pad(pn,I,F)$ is a probabilistic algorithm to implement optimized padding. It takes pn, I and F as input, determines positions j of dummy files of each keyword w_j by pseudorandom function, appends dummy files df_i to F and set $I[i,j]=1$. It outputs the random incidence matrix I' which after optimized padding.

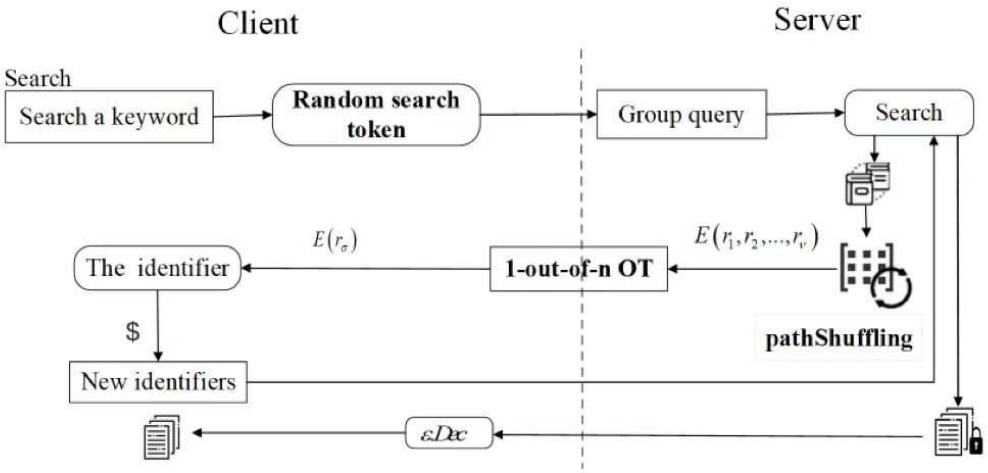

Fig. 3 Hidden search pattern diagram

Explicit search pattern hiding 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 incidence is constructed where the pseudo-random function determines the location of keywords and files 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 explicit search pattern. 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 keyword's position, the shuffling also converts the deterministic search token into a random one.

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 v according to the search token, and then retrieves the incidence matrix according to the row number group to obtain a data block of size v. 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.

After the search, the incidence matrix needs to be shuffled with the shuffling algorithm to generate the random search token. The diagram of the hidden explicit search pattern is shown in Fig. 3.

4 Details of the Proposed OSM-DSSE Scheme

4.1 Data structure

- 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 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 $\varepsilon(Gen,Enc,Dec)$.
- 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 $s_{id_j}\leftarrow G_{k_2}\left(id_j\right)$ is the key and G is a pseudo-random function; s_{id_i} indicates a file with an identifier id_i, and the column incidence $y_j\in\{1,....,n\}$ is equal to the position of s_{id_j} in T_f; *st* means status, and *st* = 1 means add while st = 0 means delete.
  2) Keyword hash table. $T_w\left(s_{w_i},x_i\right)$, where $s_{w_i}\leftarrow G_{k_2}\left(w_i\right)$ is the key, and w_i indicates the keyword; the row incidence $x_i\in\{1,....,m\}$ is equal to the position of s_{w_i} in T_w.
- Dictionary D. The dictionary stores key-value pairs $D[w]=x$, recording keywords and the corresponding line numbers.
- Two-level map $\Omega(Mw,A)$. It consists of an address map table and an array. For the address map table $Mw(l,i)$, the server searches the array incidence 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.
- Address map table $Mf(s_{id},j)$. 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

This section defines the OSM-DSSE scheme, consisting of the four algorithms $(Setup,Search,pathShuffing,Update)$ presented in the subsections below.

- $(K,I,C)\leftarrow Init(\kappa,\sigma,F)$: 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 incidence.

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

Thirdly, 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 $\left(1,2,...,\lceil m/\nu\rceil\right)$. The client constructs a two-level map $\Omega(Mw,A)$ and encrypts the key of Mw. Simultaneously, the client constructs the dictionary D according to the $T_w\left(s_{w_i},x_i\right)$ and the M_f according to the $T_f\left(s_{id_j},\langle y_j,st\rangle\right)$.

Then, the client encrypts the file by $\varepsilon.Enc$, sending 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.

Lastly, according to the optimized padding method, we execute to hide search pattern. As shown in line 15 of Algorithm 1, we determine the length of the result set that needs to be padded for each keyword $L=(L_1,\ldots,L_m)$ according to equation 3.2, select the maximum padding length as the number of dummy files, and the columns corresponding to the dummy files are also randomly inserted into the incidence matrix. Consequently, if the keyword of $i-th$ line whose response length need to be padded, we will randomly select l dummy columns $(c_1,\ldots,c_l)$ according to the padding length l and set $\{I[i,c_1]=1,\ldots,I[i,c_2]=1\}$. The positions of each keyword, file and virtual dummy in the incidence matrix are determined by the pseudorandom function G and the hash tables T_f and T_w.

Algorithm 1: $(K,I',C)\leftarrow Init(\kappa,F)$

Input: security parameter k, file set F
Output: keyword set W, random incidence matrix I, encrypted file C
Client:

1. $(K)\leftarrow DSSE.KeyGen(1^{\kappa})$
2. Extract $W=(w_1,\ldots,w_m)$ from $F=\{f_{id_1},\ldots,f_{id_n}\}$
3. for $i\in(1,\ldots,m)$ do
4. $s_{w_i}\leftarrow G_{k_2}(w_i), x_i\leftarrow T_w(s_{w_i})$
5. $(I)\leftarrow DSSE.BuildIndex(K_I,(F,W))$
6. //divide the keyword set W into data blocks of size v
7. $(W_1,W_2,\ldots,W_{m/v})\leftarrow W$
8. construct $T_w(s_{w_i},x_i)$ and $T_f(s_{id_j},\langle y_j,st\rangle)$
9. //construct Ω according to T_w
10. $\Omega(Mw,A)\leftarrow T_w(s_{w_i},x_i)$
11. //construct M_f according to T_f
12. $M_f(s_{id},j)\leftarrow T_f(s_{id_j},\langle y_j,st\rangle)$
13. $C\leftarrow\varepsilon.Enc(F)$
14. send I, C, Ω and M_f to server

Server:

15. //calculate the length of result set of each keyword
16. $L\leftarrow OptPad.ResNum(I)$
17. // cluster keywords according to their response length
18. $\Gamma\leftarrow OptPad.Cluster(L)$
19. //calculate the minimum ratio of the inserted dummy image to the real image $\gamma_{\min}$ according to 3.2
20. $\gamma_{\min}\leftarrow(\sum_{i=1}^{m}|G_i|e^r_{\max(i)})-1$
21. //calculate the expected observed probability of each cluster according to $\gamma_{\min}$
22. $\tilde{e}_i\leftarrow\frac{e^r_{\max(i)}}{1+\gamma_{\min}}$
23. // calculate the number of dummy files of each keyword according to $\tilde{e}_i$
24. $pn\leftarrow OptPad.PadNum(\Gamma)$
25. // implement optimized padding
26. $I_p\leftarrow OptPad.Pad(pn,I,F)$
27. $I'[i,j]\leftarrow I[i,j]\oplus H(K_I\|r_i)$

- $\Upsilon\leftarrow Srch(\tau)$: 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 choice to implement the 1-out-of-n OT protocol. 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 $B = (b_1, .., b_v)$ of size v, which is symmetrically encrypted.

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 v. The server calculates $((\alpha_1, c_1), ..., (\alpha_v, c_v))$, where $\alpha_j = g^{k_j} \bmod p$, $c_j = \varepsilon(m_j)\left(y / h^j\right)^{k_j} \bmod p$, $(k_j \in_R Z_p^*, j = 1, ..., v)$; α_v is the auxiliary parameter, and c_v 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.

The server returns the file sets containing dummy items to the client based on the result vector after padding. After getting the encrypted data, the client decrypts the data using the key K_F to obtain the data that satisfies the query.

Algorithm 2: $\Upsilon \leftarrow Srch(K, W)$

Input: key K, keyword set W
Output: search result Υ
Client:
1. //generate the search token
2. $(\tau_s) \leftarrow DSSE.SrchToken(K_I, w)$
3. //get the row number
4. $x_i \leftarrow D[w_i]$
5. send x_i to Server

Server:
6. //obtain the starting position of the keyword x_i in the array
7. $r_i \leftarrow M_w(l \| k_3, i)$
8. //choose v keywords to achieve the group search
9. for $j \in \{0, \ldots, v\}$
10. row.append($r_i + j$)
11. for $j \in (r_i, \ldots, r_{i+v})$
12. $\Upsilon_i \leftarrow I'[j]$
13. //Server execute the 1-out-of-n OT protocol
14. for $j \in \{1, \ldots, v\}$
15. $a_j \leftarrow g^{k_j} \bmod p$
16. $c_j \leftarrow \varepsilon(m_j)(y / h^j)^{k_j} \bmod p$ ($k_j \in_R Z_{P'}^*, j = 1, \ldots, v$)
17. send a and c to Client

Client:
18. //Client execute the 1-out-of-n OT protocol
19. for $\sigma \in \{1, \ldots, v\}$
20. $\varepsilon(m_\sigma) \leftarrow c_\sigma / \alpha_\sigma$
21. for $i \in \{1, \ldots, v\}$
22. $\Upsilon_w \leftarrow \varepsilon(m_\sigma) \oplus H(K_I \| r_i)$
23. sent Υ_w to server

Server:
24. sent encrypted file set to Client
Client:
25. decrypt the file set and obtain the data that satisfies the query

Algorithm 3: $PathShuffling(I')$:

Input: incidence matrix I'.
Output: new incidence matrix after shuffling I''
Client:
1. constructs a confusion matrix M
2. $(pk, sk) \leftarrow PE.Gen(1^{\kappa})$
3. //encrypt confusion matrix M
4. $M' \leftarrow PE.Enc_{pk}(M)$
5. send M' to server
Server:
6. //performs homomorphic calculation between the M' and I'
7. $\% \leftarrow M' \odot I'$
Client:
8. sent sk to server
Server:
9. $I'' \leftarrow PE.Dec_{sk}(\%)$
10. send I'' to client
Client:
11. update the dictionary D according to I''

- $I'' \leftarrow PathShuffling(I')$: 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 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 %, namely $\% \leftarrow M' \odot I'$. Since the Paillier encryption satisfies semantic security, the same plaintext can generate different ciphertexts. The result is homomorphically encrypted after the homomorphic calculation is performed. 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., $I'' \leftarrow PE.Dec_{sk}(\%)$, where I'' represents the result of symmetric encryption. After shuffling, the keyword's position 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.

- $(C', I'_{upd}) \leftarrow Upda(\tau_U)$: Input the key and a file, 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 $DSSE.UpdToken$ 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 that 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 $\bar{I}$ according to the T_w before it encrypts the file by $\varepsilon.Enc$. Then, the client sends (τ_u, c') to the server. The server utilizes the token to update the incidence matrix I', the address map table Mf, and the ciphertext C.

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

Algorithm 4: $(C', I'_{upd}) \leftarrow Upda(K, f)$

Input: Key K and a file f

Output: the updated matrix I'_{upd}

Client:

1. //generate the update token
2. $(\tau_u, c_f) \leftarrow UpdToken(K, f)$
3. if $op == add$
4. choose a column j
5. //update s_{id_j} in the T_f
6. $s_{id_j} \leftarrow G_{k2}(id_j)$
7. //sets the status value to 1 in T_f
8. $T_f[s_{id_j}] \leftarrow \langle y_j, 1 \rangle$
9. extracts num keywords from the file
10. for $i \in (1,\ldots,num)$
11. //update s_{w_i} in the T_w
12. $s_{w_i} \leftarrow G_{k2}(w_i)$
13. $x_i \leftarrow T_w[s_{w_i}]$
14. //update the column matrix I' according to T_w
15. $\bar{I}[x_i] \leftarrow 1$
16. $c' \leftarrow \varepsilon.Enc(\bar{I})$
17. sent (τ_u, c') to Server
18. if $op == delete$
19. choose a column j
20. //update s_{id_j} in the T_f
21. $s_{id_j} \leftarrow G_{k2}(id_j)$
22. //sets the status value to 0 in T_f
23. $T_f[s_{id_j}] \leftarrow \langle y_j, 0 \rangle$
24. $\bar{I} \leftarrow (0)^n$
25. $c' \leftarrow \varepsilon.Enc(\bar{I})$
26. sent (τ_u, c') to Server

Server:

27. if $op == add$
28. //add $\bar{I}$ to incidence matrix I'
29. $I'[j].append(\bar{I})$
30. update address map table M_f according to τ_a
31. sent I'_{upd} to Client
32. if $op == delete$
33. //replace the j column of incidence matrix I'
34. $I'[j] \leftarrow \bar{I}$
35. update address map table M_f according to τ_a
36. sent I'_{upd} to Client

4.3 Security analysis

Some definitions and theorems are given first to prove the security of the above scheme.

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

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

column matrix $\bar{I}[j]$, 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 $S=(SimStp,SimSrch,SimUpd)$ is defined as follows.

1. $(I_{M_{Sim}}',C_{Sim})\leftarrow SimStp(N,ID,<|c_1|,...,|c_n|>)$: The simulator constructs an encrypted incidence matrix and an encrypted file with random values according to the information $(N,ID,<|c_1|,...,|c_n|>)$ leaked by the leakage function $L_{stp}(I',C)$. Then, the encrypted incidence matrix and file are sent to adversary A.
2. $\tau_{b,k}^{s}\leftarrow SimSrch(l)$: The simulator randomly selects a keyword to simulate the search token $\tau_{b,k}^{s}$ according to the block number information l leaked by the leakage function $L_{srch}(I',D,Q)$. Then, the token is sent to adversary A for the search.
3. $(\tau_{fb,k}^{s},c_i)\leftarrow SimUpd(j,|c_i|)$: The simulator randomly selects a file based on the information $(j,|c_i|)$ leaked by the leakage function $L_{upd}(I_M',I_T,I[i],c_i)$ to simulate the update token $\tau_{fb,k}^{s}$ and the ciphertext file c_i. Then, the token and the ciphertext file are sent to adversary A for the update.

Theorem 1: Adaptive Semantic Security. Suppose that $\Sigma_{OSM}=(Setup,Search,Update)$ is an interactive scheme 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:

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

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 $Real_{\mathrm{A}}^{\Sigma_{OSM}}(\lambda)$ and $Ideal_{\mathrm{A},S,L}^{\Sigma_{OSM}}(\lambda)$ given in Theorem 1 is negligible. 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 $\Pr[Game_i=1]$ 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.

5 Experiment and analysis

5.1 Performance analysis

5.1.1 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., $O(m)$ and $O(n)$, 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., $O(m)$. Therefore, the total storage cost of the client is $O(m+n)$.

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

5.1.2 Communication overhead

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

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

5.1.3 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 $O(m)$.

Server computational overhead. The server needs to re-encrypt data of size v and performs $2v$ 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 $O(m^3)$.

5.2 Experimental evaluation

5.2.1 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 [42] (mail dataset) is taken as the experiment dataset. The dataset contains data from approximately 150 users, and the corpus contains 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 [43], 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).

5.2.2 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 incidence of construction time. As shown in Fig. 4, 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 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 between the search, update time, and the size of the incidence matrix is mainly investigated.

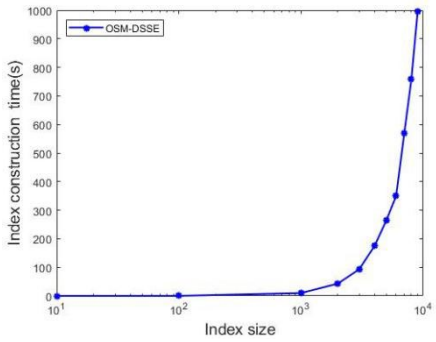

Fig. 4 The relation between token generation and number of key-word pairs

Then the cluster distance threshold is evaluated. For setup operation, keywords are clustered by it's response length into various sets, the cluster distance threshold not only affects the cluster number but also influences the maximum padding length. Therefore, it is essential to choose a suitable cluster distance threshold.

For the cluster distance threshold, a multiple of 50 distance in [50,500] is selected as the experimental data to evaluate the cluster number. The number of cluster represents kinds of responds length, excess cluster will increase time cost of clustering algorithm and matrix construction in the setup operation. It can be seen from Fig. 5 that the cluster number decreases along with the cluster distance threshold increases. When the cluster distance threshold is 50 distance with 156 clusters and when cluster distance threshold is 100 distance with 107 clusters. However, the cluster number curve tends to be flat when the cluster distance threshold exceeds 200 distance with 74 clusters, when the cluster distance threshold is 250 distance with 63 clusters.

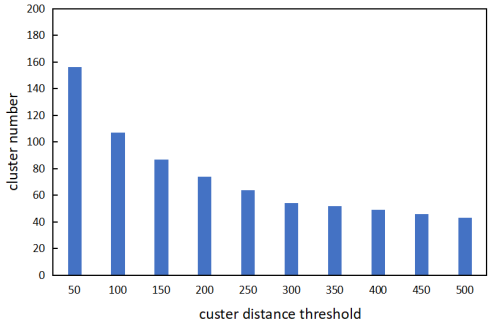

Fig. 5 The relation between cluster number and cluster distance threshold

The maximum padding length is evaluated. As shown in Fig. 6, the maximum padding length is 99, 149, and 199 for cluster numbers with the size of 100, 150, and 200, respectively.

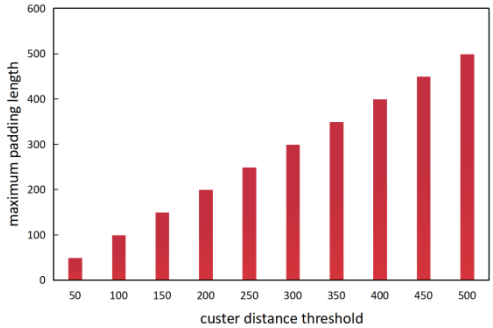

Fig. 6 The relation between cluster distance threshold and maximum padding length

To obtain the cluster distance threshold that contributes to both good cluster number and maximum padding length. Fig. 7 shows the relationship between cluster number and maximum padding length under different cluster distance threshold, where the red bar and the blue bar represent the cluster number and maximum padding length, respectively. It can be seen that the difference between the cluster number and the maximum padding length exhibits less variation for cluster distance of 200 distance and 250 distance. Since this paper's main purpose is to reduce server storage

overhead, cluster distance of 200 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.

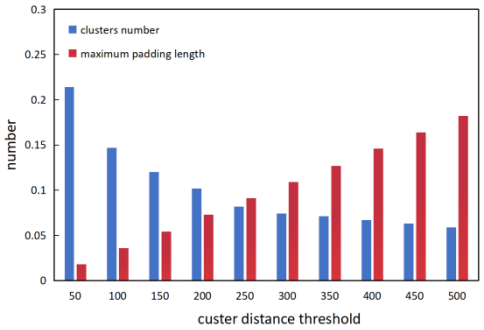

Fig. 7 The comparison between clusters number and maximum padding length

As shown in Table 1, the proposed scheme OSM-DSSE is compared with some existing 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. Only the size of the encrypted incidence is considered for server-side storage. m and n respectively denote the number of keywords and files with a maximum value of M and N, i.e., $m \le M$, $n \le N$. k represents the number of servers, ν represents the size of the data block, p represents the number of processors, z is the size of the bucket in PathORAM.

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. The DORY splits trust between multiple servers and utilize Bloom filter to hide the access pattern. Compared with the above schemes, the proposed scheme not only achieves a low storage and communication overhead, but also hides the search pattern and access pattern.

Table 1 Comparison of the proposed 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[44]	$O(m+n)$	$O(m \cdot n)$	$O\left(\frac{n}{p}\right)$	$O(m)$	×	×
DORY[16]	O(m+n)	$O(m \cdot n)$	$O(\nu)$	$O(m)$	×	√
ODSE [19]	$O((M+N)(\log N+1))$	$O(kM \cdot N)$	$O(M+N)$		×	√
DOD-DSSE[18]	$O(N)$	$O(kN^2)$	$O(N)$		√	√

Finally, the performance of search and update operations of the proposed scheme is compared with that of DOD-DSSE [18], ODSE [19], DORY [16] and Wang et al.'s scheme [17] under different incidence matrix sizes, and the results are shown in Fig. 8 and Fig. 9. The DOD-DSSE leverages two non-colluding servers and exploits the properties of an incidence matrix to avoid information leakages. The Wang et al.'s scheme uses multiplicative homomorphism and random polynomials with an appropriate degree to guarantee that the user cannot learn anything other than the desired search result. It should 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. 8 that the search time of OSM-DSSE, DORY, DOD-DSSE and Wang et al.'s scheme is 1.67s, 1.46s, 1.73s, and 3.23s respectively for the keyword-document pair with the size of 10^8. The search time of OSM-DSSE, DOD-DSSE, Wang et al.'s scheme and ODS-Tree is 1.89s, 3.47s, 2.21s, 6.11s, respectively, for the keyword-document pair with the size of 10^{10} . When the size of the keyword-document pair increases is 9×10^{10} , the search time of the Wang et al.'s scheme is approximately 4x compared to that of the OSM-DSSE. This is due to the fact that OSM-DSSE only requires searches on hash table (e.g., get keyword w_i for dictionary D) and utilize only one server, while Wang et al.'s scheme works on use of multiple servers results extra computation and communication overhead.

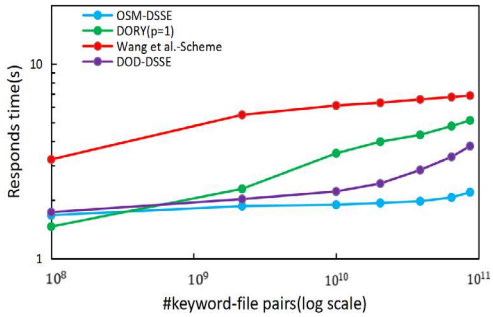

Fig. 8 The relation between search time and number of keyword-file pairs

For the update operation, it can be seen from Fig. 9 that the update time of OSM-DSSE, DOD-DSSE, $ODSE_{wor}^{wo}$ and $ODSE_{it}^{wo}$ is 1.21 s, 1.87 s, 3.17 s, and 3.97 s, respectively, for the keyword-document pair with the size of 10^8. The update time of OSM-DSSE, DOD-DSSE, $ODSE_{wor}^{wo}$ and $ODSE_{it}^{wo}$ is 1.83 s, 2.29 s, 3.63 s, and 4.83 s, respectively, for the keyword-document pair with the size of 10^8.When the size of the keyword-document pair increases to $9x10^{10}$, the update time of the $ODSE_{it}^{wo}$ is approximately 3x compared to that of the OSM-DSSE. This is because the size of update token is minimal (e.g., a binary string) while $ODSE_{it}^{wo}$ needs to transmit a lot of data to several servers.

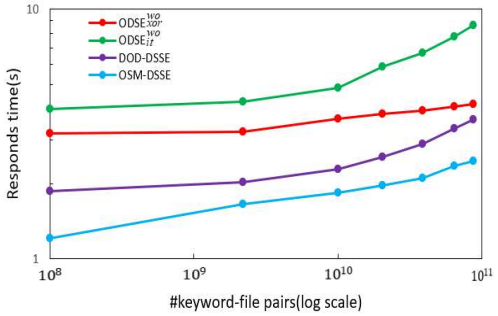

Fig. 9 The relation between update time and number of keyword-file pairs

7 Conclusion

This article proposes a searchable encryption scheme named OSM-DSSE to hide the search pattern and access pattern. 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 optimized padding strategy to realize random data access. Besides, the security of the proposed scheme is formally analyzed, showing that the proposed scheme provides adaptive semantic security that can be against selective adversaries. Furthermore, the OSM-DSSE achieves approximately 3-4x execution speed than existing schemes. The optimal block size will be investigated in the future, and the scenarios with different security levels will be updated.

Acknowledgment

The authors thank TopEdit (www.topeditsci.com) for its linguistic assistance during the preparation of this manuscript.

References

[1] D. X. Song, D. Wagner, and A. Perrig: Practical techniques for searches on encrypted data. In Proceeding 2000 IEEE Symposium on Security and Privacy, IEEE, 44–55 (2000)

[2] E.-J. Goh et al: Secure incidences. In: IACR Cryptol, ePrint Arch., 216 (2003)

[3] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky: Searchable symmetric encryption: Improved definitions and efficient constructions. Journal of Computer Security. 19, 895-934 (2006)

[4] S. Kamara, C. Papamanthou, and T. Roeder: Dynamic searchable symmetric encryption. In:

Proceedings of the 2012 ACM conference on Computer and communications security. pp. 965–976 (2012)
[5] M. S. Islam, M. Kuzu, and M. Kantarcioglu: Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: Ndss, vol. 20. Citeseer, p. 12 (2012)
[6] C. Liu, L. Zhu, M. Wang, and Y.-A. Tan: Search pattern leakage in searchable encryption: Attacks and new construction. Information Sciences. 265, 176–188 (2014)
[7] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart: Leakage-abuse attacks against searchable encryption. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, pp. 668–679 (2015)
[8] Y. Zhang, J. Katz, and C. Papamanthou: All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: 25th {USENIX} Security Symposium ({USENIX} Security 16), pp. 707–720 (2016)
[9] E. Stefanov, C. Papamanthou, and E. Shi: Practical dynamic searchable encryption with small leakage. In: NDSS, pp. 72–75 (2014)
[10] R. Bost, B. Minaud, and O. Ohrimenko: Forward and backward private searchable encryption from constrained cryptographic primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1465–1482 (2017)
[11] S.-F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal: Practical backward-secure searchable encryption from symmetric puncturable encryption. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 763–780 (2018)
[12] O. Goldreich and R. Ostrovsky: Software protection and simulation on oblivious rams. Journal of the ACM (JACM), 43(3), 431–473 (1996)
[13] J. Zhang, Q. Ma, W. Zhang, and D. Qiao: Tskt-oram: A two-server k-ary tree oram for access pattern protection in cloud storage. In: MILCOM 2016-2016 IEEE Military Communications Conference, IEEE, pp. 527–532 (2016)
[14] S. Garg, P. Mohassel, and C. Papamanthou: Tworam: Round-optimal oblivious ram with applications to searchable encryption. IACR Cryptol. ePrint Arch. 2015, 1010 (2015)
[15] S. Kamara, T. Moataz, and O. Ohrimenko: Structured encryption and leakage suppression. In: Annual International Cryptology Conference, Springer, pp. 339–370 (2018)
[16] Dauterman E, Feng E, Luo E, et al: {DORY}: An encrypted search system with distributed trust. In:14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), pp. 1101-1119 (2020)
[17] Wang Y, Sun S F, Wang J, et al: Achieving searchable encryption scheme with search pattern hidden. IEEE Transactions on Services Computing. 15,1012-1025 (2020)
[18] T. Hoang, A. A. Yavuz, and J. Guajardo: Practical and secure dynamic searchable encryption via oblivious access on distributed data structure. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, pp. 302–313 (2016)
[19] T. Hoang, A. A. Yavuz, F. B. Durak, and J. Guajardo: Oblivious dynamic searchable encryption on distributed cloud systems. In: IFIP Annual Conference on Data and Applications Security and Privacy, Springer, pp. 113–130 (2018)
[20] A. Akavia, C. Gentry, S. Halevi, and M. Leibovich: Setup-free secure search on encrypted data: Faster and post-processing free. Proceedings on Privacy Enhancing Technologies. 2019(3), 87–107 (2019)
[21] Oya S, Kerschbaum F.: Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 127-142 (2021)
[22] P. Paillier: Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques, Springer, pp. 223–238 (1999)
[23] Z. Zhang, K. Wang, W. Lin, A. W.-C. Fu, and R. C.-W. Wong: Repeatable oblivious shuffling of large outsourced data blocks In: Proceedings of the ACM Symposium on Cloud Computing, pp. 287–298 (2019)
[24] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou: Secure ranked keyword search over encrypted cloud data. In: 2010 IEEE 30th international conference on distributed computing systems, IEEE, pp. 253–262 (2010)
[25] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li: Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, pp.

71–82 (2013)
[26] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou: Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Transactions on parallel and distributed systems. 25(1), 222–233 (2013)
[27] R. Zhou, X. Zhang, X. Du, X. Wang, G. Yang, and M. Guizani: File-centric multi-key aggregate keyword searchable encryption for industrial internet of things. IEEE Transactions on Industrial Informatics, 14(8), 3648–3658 (2018)
[28] H. Cui, R. H. Deng, J. Lai, X. Yi, and S. Nepal: An efficient and expressive ciphertext-policy attribute-based encryption scheme with partially hidden access structures, revisited. Computer Networks. 133, 157–165 (2018)
[29] Z. Xia, X. Wang, X. Sun, and Q. Wang: A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE Transactions on Parallel and Distributed Systems. 27(2), 340–352 (2016)
[30] H. Cui, Z. Wan, R. Deng, G. Wang, and Y. Li: Efficient and expressive keyword search over encrypted data in the cloud. IEEE Transactions on Dependable and Secure Computing. 15(3), 409–422 (2018)
[31] G. Asharov, M. Naor, G. Segev, and I. Shahaf: Searchable symmetric encryption: optimal locality in linear space via twodimensional balanced allocations. In: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, pp. 1101–1114 (2016)
[32] I. Demertzis and C. Papamanthou: Fast searchable encryption with tunable locality. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 1053–1067 (2017)
[33] R. Bost: oφoς: Forward secure searchable encryption. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1143–1154 (2016)
[34] R. Zhang, R. Xue, T. Yu, and L. Liu: Pvsae: A public verifiable searchable encryption service framework for outsourced encrypted data. In: 2016 IEEE International Conference on Web Services (ICWS), IEEE, pp. 428–435 (2016)
[35] J. Zhang, C. Song, Z. Wang, T. Yang, and W. Ma: Efficient and provable security searchable asymmetric encryption in the cloud. IEEE Access. 6, 68384–68393 (2018)
[36] C. Liu, L. Zhu, and J. Chen: Graph encryption for top-k nearest keyword search queries on cloud. IEEE Transactions on Sustainable Computing. 2(4), 371–381 (2017)
[37] S. Kamara and T. Moataz: Boolean searchable symmetric encryption with worst-case sub-linear complexity In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, pp. 94–124 (2017)
[38] X. Ge, J. Yu, C. Hu, H. Zhang, and R. Hao: Enabling efficient verifiable fuzzy keyword search over encrypted data in cloud computing. IEEE Access. 6, 45725–45739 (2018)
[39] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner: Dynamic searchable encryption in very-large databases: data structures and implementation. In: NDSS, Citeseer, pp. 23–26 (2014)
[40] M. Naveed: The fallacy of composition of oblivious ram and searchable encryption. In: IACR Cryptol, ePrint Arch., p. 668 (2015)
[41] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas: Path oram: an extremely simple oblivious ram protocol. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 299–310 (2013)
[42] W. Cohen: The cmu enron email dataset. http://www. cs. cmu. edu/˜ enron/(2016), Accessed 2006.
[43] M. F. Porter et al.: An algorithm for suffix stripping. Program, 14(3), 130–137 (1980)
[44] T. Hoang, A. A. Yavuz, and J. Guajardo: A high-security searchable encryption framework for privacy-critical cloud storage services. IACR Cryptol, ePrint Arch. , p. 1237 (2017)
[45] Demertzis I, Papadopoulos D, Papamanthou C, et al: {SEAL}: Attack Mitigation for Encrypted Databases via Adjustable Leakage In: 29th {USENIX} Security Symposium ({USENIX} Security 20), pp. 2433-2450 (2020)
[46] Chen G, Lai T H, Reiter M K, et al: Differentially private access patterns for searchable symmetric encryption In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, pp. 810-818(2018)
[47] Patel S, Persiano G, Yeo K, et al: Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 79-93 (2019)

Appendix

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

Game$_0$: it is the same as $Real_{\mathrm{A}}^{\Sigma_{OSM}}(\lambda)$. 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 client calls Algorithm 2 to generate a search token for searching. Then, the server calls Algorithm 3 to reshuffle the path and change the next access path (Game$_0$: lines 6-9). If op=update, the client calls Algorithm 4 to generate an updated token to update the server (Game$_0$: lines 10-12). At the end of the game, adversary A outputs a bit b* (Game$_0$: lines 15-16) based on previously learned transaction credentials and other information. When b = b*, the game attacks successfully, Game$_0$ outputs 1; otherwise, it outputs 0 (Game$_0$: lines 17-18). Because Game$_0$ corresponds to a real experiment, $\Pr\left[Real_{\mathrm{A}}^{\Sigma_{OSM}}(\lambda)=1\right]=\Pr\left[Game_0=1\right]$.

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

1. $\left(st_{\mathrm{A}}, D_0, D_1\right) \leftarrow \mathrm{A}\left(1^{\lambda}\right)$
2. $b \xleftarrow{\$} \{0,1\}$

// Generate key, construct an encryption incidence matrix and ciphertext collection

3. $\left(K, \delta_0, I', c_b\right) \leftarrow Setup\left(1^{\lambda}, D_b\right)$
4. for $k=1$ to q do
5. $\left(st_{\mathrm{A}}, op_k, \left(id_{0,k}, id_{1,k}\right), \left(w_{0,k}, w_{1,k}\right)\right) \leftarrow \mathrm{A}\left(1^{\lambda}, st_{\mathrm{A}}, I', c_b, \tau_{b,1}, ..., \tau_{b,k-1}\right)$
6. if $op_k == search$ do
7. $\tau_{b,k} \leftarrow SearchToken\left(w_{b,k}\right)$ // Generate a search token
8. $\left(\delta_k, Resp\left(w_{b,k}\right)\right) \leftarrow Search\left(\tau_{b,k}, \delta_{k-1}; I', c_b\right)$ // Keyword search
9. $\left(I_k'\right) \leftarrow Search.pathShuffling\left(I_{k-1}'\right)$ // Path shuffling
10. else if $op_k == update$ do
11. $\tau_{f_{b,k}} \leftarrow UpdateToken\left(K, id_{b,k}\right)$ // Generate an update token
12. $\left(\delta_k; I_k', c_{b,k}\right) \leftarrow Update\left(\tau_{f_{b,k}}, \delta_{k-1}; I_{k-1}', c_{b,k-1}\right)$
13. end if
14. end for
15. 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
16. $b^* \leftarrow \mathrm{A}\left(st_{\mathrm{A}}, \delta_k, I', c_b, \tau_b\right) S$
17. if $b^* = b$, output 1
18. otherwise output 0

Game$_1$: it is the same as Game$_0$, except that the value of the $K \leftarrow OSM - DSSE.Gen(1^{\kappa})$ is replaced with a randomly uniformly selected value in the setup phase (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, $\Pr\left[Game_1=1\right]-\Pr\left[Game_0=1\right] \le negl(\lambda)$.

$Game_1$:	$Game_2$:
1.	$\left(st_{\mathrm{A}}, D_0, D_1\right) \leftarrow \mathrm{A}\left(1^{\lambda}\right)$
2.	$b \xleftarrow{\$} \{0,1\}$
3.	$K \leftarrow \left\{k_1, k_2, k_3\right\} \xleftarrow{\$} \{0,1\}^{\kappa}$ // Random value generates key
	//construct an encryption incidence matrix and ciphertext collection
4.	$\left(\delta_0, I', c_b\right) \leftarrow Setup\left(D_b\right)$
5.	for $k=1$ to q do
6.	$\left(st_{\mathrm{A}}, op_k, \left(id_{0,k}, id_{1,k}\right), \left(w_{0,k}, w_{1,k}\right)\right) \leftarrow \mathrm{A}\left(1^{\lambda}, st_{\mathrm{A}}, I', c_b, \tau_{b,1}, ..., \tau_{b,k-1}\right)$
7.	if $op_k == search$ do
8.	$\tau'_{b,k} \leftarrow SearchToken\left(Random\left(w_{b,k}\right)\right)$ // Randomly select a keyword to generate a search token
9.	$\left(\delta_k, Resp\left(w_{b,k}\right)\right) \xleftarrow{OT_m^1} Search\left(\tau'_{b,k}, \delta_{k-1}; I', c_b\right)$

10. $\left(I_k'\right) \leftarrow Search.pathShuffling\left(I_{k-1}'\right)$
11. else if $op_k == update$ do
12. $\tau'_{f_{b,k}} \leftarrow UpdateToken\left(K, Random(\cdot)\right)$ // Generate a column matrix with random values
13. $\left(\delta_k; I_k', c_{b,k}\right) \leftarrow Update\left(\tau'_{f_{b,k}}, \delta_{k-1}; I_{k-1}', c_{b,k-1}\right)$
14. end if
15. end for
16. let $\tau_b = \left(\tau_{b,1}, ..., \tau_{b,q}\right)$
17. $b^* \leftarrow \mathrm{A}\left(st_{\mathrm{A}}, \delta_k, I', c_b, \tau_b\right)$
18. if $b^* = b$, output 1
19. otherwise output 0

Game2: it is the same as Game1, except that the determination function for generating the search or update token is replaced by a random function (Game2: lines 8, 12). 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, $\Pr\left[Game_2 = 1\right] - \Pr\left[Game_1 = 1\right] \le negl(\lambda)$.

Moreover, since the 1-out-of-n OT protocol performed in search is based on the difficult problem of DDH (Game2: 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.

Game3: it is the same as Game2, 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 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 $\Pr\left[Game_3 = 1\right] - \Pr\left[Game_2 = 1\right] \le negl(\lambda)$ is obtained.

Game4: it is the same as Game3, except that the searched block with maximum number of a keyword.

Theorem 3: Leakage function L_{srch} is volume-hiding.

Proof: We present a optimized padding method to prevent volume-hiding leakage, in our scheme we utilize cluster algorithm to sort out keywords into different sets according to their number of searched blocks, therefor the number of the searched blocks of each keyword in the same sets are equal. Since the adversary A don't has information of the number of keywords in cluster set C_i, when $\mathcal{A}$ recieves a result set with k_i searched blocks, A can not distinguish if it's a keyword searched before or other keyword in C_i. Note that the only other leakage is the number of cluster which is independent of response lengths of the query operations. Consequently adversary A can't get any information from the number of searched blocks, completing the proof.

Game4 outputs the maximum volume of each search operation. So the equation $\Pr[Game_4 = 1] - \Pr[Game_3 = 1] \le negl(\lambda)$ is obtained.

$\boxed{Game_5}$: $b \leftarrow Ideal_{A,S,L}^{\Sigma_{OSM}}(\lambda)$

1. $(st_A, D_0, D_1) \leftarrow A(1^\lambda)$
2. $b \xleftarrow{\$} \{0,1\}$

// Generate encrypted incidence and ciphertext based on leakage function

3. $\boxed{(st_S, (I_M', I_T), c_b) \leftarrow SimStp(1^\lambda, L_{stp}, D_b)}$
4. for $k = 1$ to q do
5. $(st_A, op_k, (id_{0,k}, id_{1,k}), (w_{0,k}, w_{1,k})) \leftarrow A(1^\lambda, st_A, I', c_b, t_{b,1}, ..., t_{b,k-1})$
6. if $op_k == search$ do

// Generate a search token based on leakage function to search

7. $\boxed{(\tau_{b,k}^s,) \leftarrow SearchToken(w_{b,k} \xleftarrow{\$} L_{srch}(m_l, N))}$
8. $\boxed{(st_S; Resp^s(w_{b,k})) \leftarrow SimSrch(st_S; \tau_{b,k}^s; I', c_b)}$
9. $(I_k') \leftarrow Search.pathShuffling(I_{k-1}')$
10. else if $op_k == update$ do

// Generate update token based on leakage function to update

11. $\boxed{\tau_{fb,k}^s \leftarrow UpdateToken(id_{b,k} \xleftarrow{\$} L_{upd}(ID_i, |c_i|))}$
12. $\boxed{(st_S; I_k', c_{b,k}) \leftarrow SimUpd(st_S; \tau_{fb,k}^s; I_{k-1}', c_{b,k-1})}$
13. end if
14. end for
15. let $\tau_b = (\tau_{b,1}, ..., \tau_{b,q})$
16. $b^* \leftarrow A(st_A, \delta_k, I', c_b, \tau_b)$
17. if $b^* = b$, output 1
18. otherwise output 0

$Game_5$: 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 $SimUpd(\cdot)$ ($Game_4$: lines 3, 8, 12). According to the above analysis, the output of the simulator and $Game_3$ is indistinguishable, so $\Pr[Game_5 = 1] - \Pr[Game_4 = 1] \le negl(\lambda)$ 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, including $Game_0$, $Game_1$, $Game_2$, $Game_3$, and $Game_4$, it can be obtained that $\left|\Pr\left[Real_A^{\Sigma_{OSM}}(\lambda) = 1\right] - \Pr\left[Ideal_{A,S,L}^{\Sigma_{OSM}}(\lambda) = 1\right]\right| \le negl(\lambda)$.

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