A Verifiable Symmetric Searchable Encryption Scheme based on the AVL Tree

Qing Wang[1], Xi Zhang[1], Jing Qin[1,2,*], Jixin Ma[3], Xinyi Huang[4]

[1]*School of Mathematics, Shandong University, Jinan, Shandong 250100, China*
[2]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China*
[3]*School of Computing and Mathematical Sciences University of Greenwich, London,UK*
[4] *College of Mathematics and Informatics, Fujian Normal University, Fuzhou, Fujian 350117, China*
[*]*Corresponding author: qinjing@sdu.edu.cn*

Verifiable symmetric searchable encryption is a keyword search technology that supports verification of search results. Many schemes improve search performance by dividing each keyword label into segments and storing them in a Trie-tree at the expense of high storage. And the index will degenerate into a linear linked list when all keyword labels have the same prefix except for the last segment. But it will greatly affects the search efficiency. In this paper, we propose a verifiable symmetric searchable encryption scheme based on the AVL Tree (abbreviated as VSSE-AVL), which uses complete keyword labels to build the index. Compared with the Trie-tree index, VSSE-AVL not only balances storage and search performance, but also avoids degradation. To verify the correctness and completeness of empty search results, we store path information in each leaf node and node with only one child node. Considering the substitution attack, we bind the file identifier and the file so that the client will find out once the server returns inconsistent search results. Rigorous security analysis shows VSSE-AVL satisfies privacy and verifiability. Compared with the verifiable SSE-2 with the same security, the experimental evaluation shows that our proposed scheme performs better on storage, search and verification.

Keywords: keyword search; data verifiability; AVL tree

Received 00 January 2009; revised 00 Month 2009

1. INTRODUCTION

Cloud storage [1, 2] has massive storage space, pay-on-demand and flexible scalability, so more and more clients are attracted to outsource data to cloud servers. However, the client's data may be compromised due to server failures or malicious attacks. Research shows that data breach attacks have increased 93 percent in 2020 [3].

To maintain data privacy without losing data availability, Song et al. [4] introduced the primitive of symmetric searchable encryption, or SSE for short. SSE is a keyword search technology, which allows the client to search on the ciphertext without decryption. So far, plenty of works [5–9] have been done on SSE. But they only focused on honest-but-curious servers who follow the protocol specification honestly but attempt to learn the underlying information from the index, ciphertext set and trapdoor. In more realistic scenarios, the server may also deviate from the protocol specification. To reduce computation or communication overhead, he may return partial search results or false search results. For example, in the railway passenger information database, when tracing close contacts of Covid-19, more lives are at risk if no exhaustive search conducts. In the hospital information system, doctors search for patients who need treatment, but the system randomly returns a list of patients without searching. It will cause some patients to delay treatment or unnecessary treatment.

In order to solve the above problems, Chai et al. [10] firstly proposed verifiable symmetric searchable encryption (VSSE), which ensures the searchability and verifiability of data. Verifiability is the ability to verify the correctness and completeness of search results. Correctness means that each ciphertext in the search results matches the search keyword, and completeness means that the search results contain all ciphertexts that match the search keyword. Then Kurosawa et

al. [11] introduced privacy and reliability to define the security of VSSE and proved that this security is equivalent to universally composable (UC) security. Hu et al. [12] proposed a verification mechanisms suitable for different schemes based on smart contracts. Search performance is one of the important indicators for evaluating a scheme. Many schemes [13–15] built the index based on various trees to speed up searches, but it greatly increases the storage space and does not prevent the server from searching with linear complexity when all keyword labels has the same prefix except for the last segment (a character or vector).

To reduce computation and communication overhead, a malicious server return partial search results or false search results. However, most schemes [10, 11, 13, 16] do not consider the case of empty search results. Furthermore, the schemes in [10, 13] could verify the correctness and completeness of file identifiers but did not consider the corresponding relationship between the file and its identifier. Under the semi-honest model, this problem does not need to be considered because the server honestly follows the protocol. But the malicious server may deviate from the protocol to perform substitution attacks (the returned ciphertexts do not match the returned identifiers) [16]. These schemes [16–19] solved the problem by computing the hash function value of the file content and its identifier. But they require too much calculation.

In this paper, our main goal is to design a VSSE scheme that performs well on storage, search and verification. Since the server is malicious, the scheme must also be able to verify the correctness and completeness of search results even if they are empty. For security reasons, the scheme should resist substitution attacks.

1.1. Our contributions

We propose an efficient verifiable symmetric searchable encryption scheme based on the AVL tree, abbreviated as VSSE-AVL. It solves the following three challenges in VSSE.

1. How to balance storage and search performance. In our scheme, the index (named MAVL) is an AVL tree generated from complete keyword labels, where each node represents a keyword. Compared with the schemes in [10, 13–15, 20] where each path represents a keyword, the number of nodes in MAVL is just the sum of their leaf nodes. Their search time is related to the number of segments of the keyword label. When the number is greater than $\log m$ (m denotes the number of keywords), our scheme is superior to them in terms of storage and search. When the number is less than $\log m$, we have an advantage in storage, but the search speed is not as fast as them.

2. How to verify the correctness and completeness of search results even if they are empty. In our scheme, the correctness and completeness of empty search results are the correctness of the path searched by the server. Since P^1 of the last node on the search path needs to be returned, the client can compare the trapdoor with each label in P^1 to verify whether P^1 is the search path for the search keyword. And P^2 is the pseudo-random value of P^1, so the client can verify the correctness of P^1 through P^2 to verify the correctness and completeness of empty search results. The main calculation is to verify the search path, so the verification time is small and it grows slowly with the increase of records.

3. How to resist substitution attacks. In our scheme, the encryption key for each file is generated by pseudo-random function acting on the auxiliary key and its identifier. So the file can be decrypted correctly only if the correct auxiliary key and the file identifier corresponding to the file are held. Once the server returns an inconsistent file identifier and ciphertext, the client can find out.

1.2. Related Works

In the public key setting, a lot of searchable encryption schemes [21–23] have been proposed. They leveraged public keys to build indexes so that clients do not need to negotiate keys in advance. So this method is applicable for multi-user scenarios. For efficiency, we mainly study symmetric searchable encryption.

Symmetric Searchable Encryption. The first SSE scheme was proposed in [4], its main idea is to encrypt each keyword separately, and embed the stream cipher and its pseudo-random function value into the ciphertext to build the index. When searching for a specified keyword, the server needs to traverse all ciphertexts and verify whether the value after XOR has a special format. Inevitably, the search complexity depends linearly on the size of the file.

In 2003, Goh [5] defined a security model for SSE referred to as semantic security against adaptive chosen keyword attack, and proposed an efficient SSE scheme using bloom filter under this model. However, the security model only considers the security of indexes. To make the adversary can't deduce any useful information from the index and trapdoor, Curtmola et al. defined adaptive semantic security based on simulation in [7].

Verifiable Symmetric Searchable Encryption. Chai et al. [10] proposed the first VSSE scheme which ensures the searchability and verifiability of data. The scheme built index by dividing each keyword into characters, storing them in $PPTrie$ and encrypting the nodes. Since each node contains the character information of the parent node, clients can verify the correctness and completeness of the search results. However, there are no binding file identifiers and files, so the scheme is not resistant to substitution attacks.

In 2013, Wang et al. [13] proposed a verifiable searchable encryption scheme that supports fuzzy keyword search. The client generates fuzzy keyword sets, seg-

ments the hash value of each fuzzy keyword by n bits and stores them in the symbol tree. But the scheme is also not resistant to substitution attacks. In [14], the verification is achieved by generating the hash value of the root node, but this requires returning the minimum sub-tree and partial node values. Also based on the idea of segmentation, Zhu et al. [15] proposed generic verifiable SSE based on merkle patricia tree to support updates and improve search performance at the cost of computation and storage overhead.

Kurosawa et al. [11] introduced privacy and reliability to define security, and proved that this security is equivalent to UC security. This security ensures that no adversary can obtain any useful information from the index, ciphertext set and trapdoor, and no adversary can forge search results to pass verification. Under the above security model, they proposed verifiable SSE-2 based on SSE-2 in [7]. For data flexibility, they proposed dynamic VSSE [16] based on the RSA accumulator. In [24], the client generated authentication tags for indexes based on homomorphic MAC technique in [25] to realize the verification of top-k multi-keyword search results.

Malicious servers may return empty search results, but clients cannot verify the results in the above scheme. In 2015, Taketani et al. [17] proposed ameliorate scheme based on verifiable SSE-2 in [11] which can verify the correctness and completeness of the search results even if they are empty. The client sorts the arrays in ascending order of labels, and stores the information of the previous array in each array. In this way, once the server returns the proof, the client can distinguish whether the search keyword exists in the ciphertext set. In [26, 27], the client stores the data in the Bloom filter tree or the Invert Bloom filter, and uploads it to a trusted party. With the help of the trusted party, the client can verify whether the search keyword exists in the ciphertext set.

1.3. Organization

The rest of the paper is organized as follows. In Section 2, we show the system and security model for our scheme. Following in Section 3, we briefly introduce the AVL tree, provide the specific algorithm of VSSE-AVL, prove its security under the security model mentioned in Section 2, and show an instance of VSSE-AVL. In Section 4, we provide performance evaluation of VSSE-AVL. Finally, the conclusions are given in Section 5.

2. PROBLEM FORMULATION

We introduce the system model of VSSE and define the security of VSSE under this model. In addition, we review the design goals for our scheme again.

2.1. System Model

In this paper, we consider a frequently-used scenario in VSSE. As illustrated in Fig.1, it involves three roles: a server who provides data storage and search service, a data owner who outsources file sets to the server, and a data user who searches files containing search keywords. In our system model, the server is regarded as a malicious role and the data owner and the data user are considered to be trusted. We assume that the data owner has authorized search capabilities to the data user. That is, the data owner sends the keys to the data user through a secure channel in advance. And sometimes the data owner and the data user are the same agent, who can be called the client.

To support keyword search on the ciphertext set,

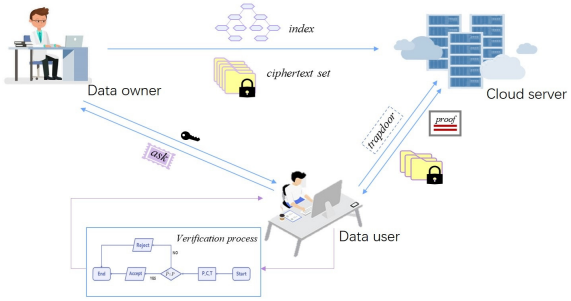


FIGURE 1. System model of MAVL

the data owner builds the index based on the keyword set extracted from the file set, and uploads the index and ciphertext set (encrypted file set) to the cloud server. Once an authorized data user wants to search files that contain a specified keyword, he generates the trapdoor and submits it to the server. Upon receiving the search request, the server searches the index and returns the matching ciphertext set to the user. Since a malicious server may return partial search results or false search results to reduce computation or communication overhead, the data user needs to verify the correctness and completeness of the search results.

2.2. Security Definition

In our system model, we consider the server to be malicious. He may try to learn any useful information about files and keywords from indexes, ciphertext sets and trapdoors. Besides, he can deviate from the protocol to return partial search results or false search results. We introduce privacy and verifiability to define security for VSSE according to the adversary's ability. To formally define privacy, we first introduce the concept of the leakage function $\mathcal{L}$.

Definition 1 (Leakage function $\mathcal{L}$)**.** Let $\mathcal{L} = (\mathcal{L}_1, \mathcal{L}_2)$. $\mathcal{L}_1$ is a leakage function that describes the information that is allowed to be learned from indexes and ciphertext sets. It consists of the number of keywords $|W|$, the size of the file $(|d_1|, |d_2|, \ldots, |d_n|)$ and the relation-

ship $D(W) = \{d(w)|w \textit{ is the search keyword}\}$ where $d(w) = \{d_i \textit{ containing } w\}_{i\in[1,n]}$. $\mathcal{L}_2$ is another leakage function that describes the information that is allowed to be learned from trapdoors. It is a symmetric binary matrix $\sigma(QW)$, where the value in the i^{th} row and j^{th} column is 1 only if $w_{i^{th}} = w_{j^{th}}$ ($w_{i^{th}}$ denotes the i^{th} search keyword).

Definition 2 ($\mathcal{L} - privacy$)**.** Suppose $VSSE_1$ be a verifiable symmetric searchable encryption scheme, λ is the security parameter, the following $\mathbf{Game_{real}}$ is a real game and played by a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$, and $\mathbf{Game_{sim}}$ is a simulation game and played by a challenger $\mathcal{C}$, an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$. We say that $VSSE_1$ is $\mathcal{L} - privacy$ if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that $|Pr(\mathcal{A} \textit{ outputs } b = 1 \textit{ in } \mathbf{Game_{real}}) - Pr(\mathcal{A} \textit{ outputs } b = 1 \textit{ in } \mathbf{Game_{sim}})| \leq negl(s)$ for any probabilistic polynomial-time adversary $\mathcal{A}$. $\mathcal{L} - privacy$ means that the adversary cannot obtain any useful information from the index, ciphertext set, and trapdoor except for leakage functions.

$\mathbf{Game_{real}}$:

- $\mathcal{C}$ runs $KeyGen(1^s)$ to generate a key set K.
- $\mathcal{A}$ chooses a document set D, extracts the keyword set W from D and sends them to $\mathcal{C}$.
- $\mathcal{C}$ runs $BuildIndex$ to generate the ciphertext set C and builds the index MAVL with (K, D, W).
- for $2 \leq i \leq q+1$,
  1. $\mathcal{A}$ chooses a keyword w_i based on $\{(C, MAVL, Tr_{w_1}, \ldots, Tr_{w_{i-1}}\}$ and sends it to $\mathcal{C}$,
  2. $\mathcal{C}$ computes the trapdoor Tr_{w_i} by calling algorithm $Trapdoor(K, w_i)$, and sends it to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b.

$\mathbf{Game_{sim}}$:

- $\mathcal{A}$ chooses (K, D, W) and sends them to $\mathcal{C}$.
- $\mathcal{C}$ sends the leakage functions $\mathcal{L}_1$ to $\mathcal{S}$.
- $\mathcal{S}$ computes the ciphertext set C and the index MAVL with $\mathcal{L}_1$.
- for $2 \leq i \leq q+1$,
  1. $\mathcal{A}$ chooses a keyword w_i based on $\{(C, MAVL, Tr_{w_1}), \ldots, Tr_{w_{i-1}}\}$ and sends it to $\mathcal{C}$;
  2. Given leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$, $\mathcal{S}$ computes the trapdoor Tr_{w_i} and sends it to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b.

Definition 3 ($Verifiability$)**.** Suppose $VSSE_1$ be a verifiable symmetric searchable encryption scheme. The following **Game** is played by a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. We say that $VSSE_1$ is $Verifiability$ if the probability of any probabilistic polynomial-time adversary $\mathcal{A}$ winning the game is negligible for any (D, W) and any search keyword w. $Verifiability$ means that the adversary cannot forge the search results to pass the verification algorithm.

Game :

- $\mathcal{C}$ chooses (K, D, W), runs $BuildIndex$ and sends $\{C, MAVL\}$ to $\mathcal{A}$.
- for $1 \leq i \leq q$,
  1. $\mathcal{C}$ chooses a keyword w_i adaptively, and compute trapdoor Tr_{w_i} by $Trapdoor(K, w_i)$ for $\mathcal{A}$.
  2. $\mathcal{A}$ searches on MAVL, and returns the matching ciphertext set and proof set $(CSet^*, PSet^*)$ to $\mathcal{C}$.
- $\mathcal{C}$ runs $Verification(K, CSet^*, PSet^*)$ and outputs $accept$ or $reject$.
  We say that $\mathcal{A}$ wins if $CSet^* \neq CSet$ but $Verification(K, CSet^*, PSet^*) = accept$.

2.3. Design Goals

To safely and efficiently outsource data to the cloud server, our scheme should achieve the following goals:
1) The scheme should be feasible for both storage and search.
2) The data user can verify the correctness and completeness of the search results even if they are empty.
3) The scheme should be able to resist substitution attacks.

3. VSSE-AVL

To make it easier to understand the scheme, we first review the concept of AVL tree, and then present concrete construction and security analysis of our scheme. For the sake of clarity, an example of our scheme is shown in Fig.2.

3.1. AVL Tree

The AVL tree is proposed by Adelse-Velskil et al. in [28], which is a rooted binary search tree. Each node stores a label and has a left subtree and a right subtree. The label in each node must be greater than all labels stored in the left subtree, and not greater than any label in the right subtree. Therefore, we can obtain ordered sequence of labels by inorder traversal of the AVL tree. Furthermore, the AVL tree is a highly self-balancing binary tree, in which the difference between the height of two subtrees of each node is at most 1. So it will not degenerate into a linear linked list. An instance of AVL tree is shown in Fig.2 (Only consider the label and ignore other attributes of nodes).

To build an AVL tree from label set, we first insert the first label as the root node. If the subsequent label is less than the label of the root node, the new node is recursively inserted into the left subtree; if the label is greater than the root, it is inserted into the right subtree. After each insert operation, rebalance need to be done to keep balance of the AVL tree. To search a given label in the AVL tree, one begins with the root node and sets the current node as root node. If the current node is null, the AVL tree is empty. If the label equals that of the current node, the search is successful. If the label is larger than that of the current node, search from its right subtree and set current node as the root node of right subtree, otherwise search from its left subtree. The process has been repeated until the label is matched or a path has been traversed.

Similar to the AVL tree, the Red–black tree [29] is self-balancing binary search tree. But searching on AVL trees is much faster than Red–black tree because they are more strictly balanced. Compared with the Trie tree, the number of nodes in AVL tree is just the sum of their leaf nodes.

3.2. Concrete construction

Suppose that $\Sigma = (Gen(\cdot), Enc_{sk}(\cdot), Dec_{sk}(\cdot))$ is a secure symmetric encryption scheme with indistinguishability against chosen-plaintext attacks (IND-CPA), $f_k : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is a pseudo-random function, $D = \{d_1, d_2, \ldots, d_n\}$ and $W = \{w_1, w_2, \ldots, w_m\}$ are the file set and keyword set. Let i denotes file identifier of d_i.

• $KeyGen(1^s)$: a probabilistic algorithm run by the data owner, he chooses a pseudo-random function $f(\cdot,\cdot) : \{0,1\}^\star \times \{0,1\}^s \rightarrow \{0,1\}^s$. And he generates a key set $K = \left\{k^{'}, k^{''}, k^{'''}, k\right\}$ from $\{0,1\}^s$, where k is an auxiliary key. For convenient, we use $f_{k'}(\cdot)$ instead of $f(\cdot, k^{'})$, and the same is true for others.

• $BuildIndex(D, W, K)$: a probabilistic algorithm run by the data owner to build the index. It takes D, W, K as inputs.

1) To bind a file and its identifier, the data owner computes key $k_i = f_k(i), i \in [1, n]$ for d_i, then invokes encryption algorithm Σ to generate ciphertext $c_i = \Sigma.Enc(k_i, (d_i)), i \in [1, n]$.

2) For $w \in W$, the data owner calculates the label $L_w = f_{k'}(w)$ and creates the array A_w to store all file identifiers in $d(w) = \{i_1, \ldots, i_s\}$. For the sake of completeness verification, the corresponding proof $P_w = f_{k''}(w \parallel i_1 \parallel \cdots \parallel i_s)$ need to be stored.

3) Based on the label set $L = \{L_{w_1}, L_{w_2}, \ldots, L_{w_m}\}$, the data owner builds an AVL tree named MAVL, where each node contains three attributes L_w, A_w, P_w. To verify empty search results, apart from the above-mentioned attributes, each leaf node and node with only one child node needs to contain two additional attributes P_w^1 and P_w^2. P_w^1 represents the value and the pseudo-random function value of all the node labels from the root node to the leaf node in an orderly concatenation and $P_w^2 = f_{k'''}(P_w^1)$. There is a description of the additional attributes of the nodes in the instance in Section 3.4.

4) The index MAVL, ciphertext set $C = \{(1, c_1), (2, c_2), \ldots, (n, c_n)\}$ are uploaded to the server.

• $Trapdoor(k^{'}, w)$: a deterministic algorithm run by the data user, and it takes the search keyword w and key $k^{'}$ as inputs. The trapdoor $Tr_w = f_{k'}(w)$ is computed and submitted to the server.

• $SearchIndex(MAVL, Tr_w)$: is a deterministic algorithm run by the server to search matching ciphertexts. It takes Tr_w, MAVL and C as inputs.

Upon receiving Tr_w, the server searches over MAVL using Algorithm 1 and returns matching ciphertext and proof set. Specifically, the server begins with the root node and sets the current node as root node. If the current node is null, the search keyword does not exist in the ciphertext set. If the trapdoor Tr_w equals the label of the current node, the server returns the matching ciphertext set $Cset = \{(j_1, c_{j_1}), \ldots, (j_k, c_{j_k})\}$ and the proof set $PSet = \{0, cn.P\}$ (P is the proof mentioned above for completeness verification) and terminates the search. If the trapdoor is less than the label of the current node, the server searches the left subtree. Otherwise, the server searches the right subtree. The process has been repeated until the server returns search results or a path has been traversed. If MAVL is traversed but no label equal to the trapdoor, let $CSet = \emptyset$ and $PSet = \left\{1, lastnode.P^1, lastnode.P^2\right\}$ ($lastnode$ denotes the last node on the search path, and P^1, P^2 are the two additional attributes mentioned above) and return them to the data user.

Algorithm 1 Search algorithm on the MAVL

Input: The index MAVL and The trapdoor Tr_w
Output: The ciphertext set $CSet$ and the proof set $PSet$

```
Set current-node (cn) as root node of MAVL
if cn is not empty then
    if Tr_w == cn.Label then
        Find cn's identifiers is based on cn.A;
        Find cn's ciphertexts based on is;
        Append (i, ciphertext)s to CSet;
        Append 0, cn.P to PSet;
        break;
    else if Tr_w <= cn.Label then
        Search algorithm on the cn's left subtree
    else if Tr_w >= cn.Label then
        Search algorithm on the cn's right subtree
    else
        Append 1, lastnode.P^1, lastnode.P^2 to PSet;
    end if
end if
```

• $Verification(K, CSet, PSet)$: a deterministic algorithm run by the data user to verify the correctness and completeness of the search results. It takes K, $CSet$ and $PSet$ as inputs . It is divided into the following two cases according to whether the $PSet$ contains 0 or 1:

The $PSet$ contains 0 that means there are some files contain the search keyword. The data user need to verify the correctness and completeness of $Cset = \{(j_1, c_{j_1}), \ldots, (j_k, c_{j_k})\}$ to prevent the server from returning partial or false search results. The details of verification are as follows:

*Step*1 : The step is to check the correctness and completeness of file identifiers, the data user computes the $f_{k''}(w \parallel j_1 \parallel \cdots \parallel j_k)$ and compares it with P. If the equality holds, all identifiers of file containing w have been returned, next go to *Step*2. Otherwise, terminate the process of verification and output *reject*.

*Step*2 : The step is to check the correspondence between the file identifier and the ciphertext. The data user computes the key $k_j = f_k(j), j \in Cset$ for c_j. If all ciphertexts in $CSet$ are validly decrypted, the data user convinces of the server is honest and outputs *accept*.

If $PSet$ contains 1, no matching label is found after travelling MAVL. The data user also verifies the correctness and completeness of empty set. The details are as follows:

*Step*1 : The data user first verifies the correctness of search path, that is, the correctness of $lastnode.P^1$. Assume that $lastnode.P^1 = a_1 \parallel \cdots \parallel \ a_k$, if $(Tr_w - a_i)(a_{i+1} - a_i) \geq 0, i \in [1, k-1]$ holds, go to *Step*2. Otherwise, output *reject*.

*Step*2 : The data user also computes the pseudo-random function value of $lastnode.P^1$ and compares it with $lastnode.P^2$. If it is equal, no files in the database contain the search keyword w, the data owner outputs *accept*.

3.3. SECURITY ANALYSIS

Theorem 3.1. *The above scheme is $\mathcal{L}-$privacy if Σ is an IND-CPA secure symmetric encryption scheme and f is a pseudo-random function.*

Proof. To prove the server cannot learn any useful information about files and keywords from the index, ciphertext set and trapdoor, we will show that there exists a probabilistic polynomial-time simulator $\mathcal{S}$, such that $\mathbf{Game_{real}}$ and $\mathbf{Game_{sim}}$ are computationally indistinguishable for any probabilistic polynomial-time adversary $\mathcal{A}$. First, we construct a simulator as below.

Given $\mathcal{L}_1 = \{(|d_1|, |d_2|, \ldots, |d_n|), |W| = m, D(W)\}$, $\mathcal{S}$ runs $KeyGen(1^s)$ to obtain the key set $K = \left\{k, k', k'', k'''\right\}$ and chooses m random strings as keywords $w_{s,i}, i \in [1, m]$. Meanwhile, $\mathcal{S}$ simulates each ciphertext by computing $c_{s,i} = \Sigma.Enc(f_k(i), 0^{|d_i|}), i = 1, \ldots, n$. For $w_{s,i}, i \in [1, m]$, $\mathcal{S}$ computes the label and creates an array. If $d(w_i) \in D(W)$, $w_{s,i}$'s array is used to store file identifiers in $d(w_i)$. Otherwise, the array is filled with some random file identifiers. And $MAVL'$ is built based on the label set $L = \left\{L_{w_{s,1}}, L_{w_{s,2}}, \ldots, L_{w_{s,m}}\right\}$ where each non-leaf node contains three attributes (L, A, P), and each leaf node and node with only one child node contains two additional attributes P^1, P^2.

Given $\mathcal{L}_2 = \sigma(QW)$, $\mathcal{S}$ simulates the trapdoor for the search keyword. During the j^{th} search, the adversary searches files containing the keyword $w_{j^{th}}$. If there exists $w_{i^{th}}, i < j$ such that $w_{j^{th}} = w_{i^{th}}$, $\mathcal{S}$ sets $Tr_{w_{s,j}} = Tr_{w_{s,i}}$. Otherwise, $\mathcal{S}$ chooses $w_{s,j}$ but not used before, computes its pseudo-random function value and sets it as the trapdoor.

We use $\mathbf{Game_0}$ and $\mathbf{Game_2}$ to represent $\mathbf{Game_{real}}$ and $\mathbf{Game_{sim}}$ respectively. And let p_i denotes the probability that $\mathcal{A}$ outputs 1 in $\mathbf{Game_i}$. Therefore, we need prove that any probabilistic polynomial-time $\mathcal{A}$ cannot distinguish between $\mathbf{Game_0}$ and $\mathbf{Game_2}$ with non-negligible advantage.

We define a modified game, called $\mathbf{Game_1}$. $\mathbf{Game_1}$'s structure and $\mathbf{Game_0}$ are the same but the ciphertext set in the $\mathbf{Game_1}$ is simulated by $\mathcal{S}$. Since Σ is IND-CPA, $|p_0 - p_1| < negl(s)$. $\mathbf{Game_2}$ is nearly identical to $\mathbf{Game_1}$, the only difference is that the index replaced by $MAVL'$. Since all probabilistic polynomial-time adversaries cannot distinguish the output of the pseudo-random function from the random number, and the file identifiers stored in all arrays in the $MAVL'$ obtained according to the leakage function or is generated randomly, $|p_1 - p_2| < negl(s)$. Therefore, we obtain $|p_0 - p_2| < negl(s)$. That is any probabilistic polynomial-time $\mathcal{A}$ cannot distinguish between $\mathbf{Game_0}$ and $\mathbf{Game_2}$ with non-negligible advantage. □

Theorem 3.2. *The above scheme is $Verifiability$ if f is a pseudo-random function.*

Proof. We will prove any probabilistic polynomial-time $\mathcal{A}$ wins **Game** with negligible probability. In other words, for any search keyword w, the probability that $\mathcal{A}$ returns $CSet_w^* \neq CSet_w$ and $\mathcal{C}$ outputs *accept* is negligible.

Suppose that there exists a probabilistic polynomial-time $\mathcal{A}$ who breaks the $Verifiability$ of above scheme with a non-negligible probability. Namely, there is at least one w that makes the $\mathcal{A}$ to forge $CSet_w^* = \{(1, c_1^*), (2, c_2^*), \ldots, (s, c_s^*)\}$ which is not equal to $CSet_w = \{(1, c_1), (2, c_2), \ldots, (s, c_s\}$, but it is accepted by the $\mathcal{C}$ with non-negligible probability. This requires $\mathcal{A}$ to be able to calculate the pseudo-random function value of file identifiers or search paths. But it goes against the collision resistance property of pseudo-random function. Therefore, any probabilistic polynomial-time $\mathcal{A}$ can not break the $Verifiability$ of the scheme with non-negligible probability.

□

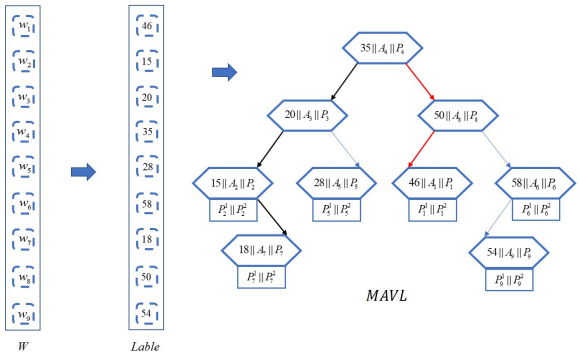


FIGURE 2. MAVL

3.4. AN INSTANCE OF VSSE-AVL

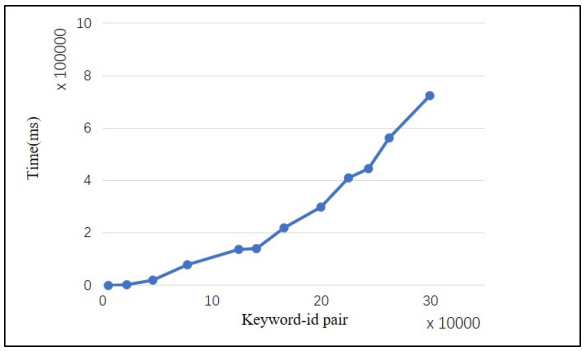
To understand the scheme more easily, we provide an instance of VSSE-AVL. Its index is shown in Fig.2. There are 9 keywords: $w_1, w_2 \ldots, w_9$, and their labels correspond to $46, \ldots, 54$ respectively. Based on these labels, an AVL tree is built, where each node contains three attributes: label,A_w, P_w. To verify empty search results, each leaf node and node with only one child node needs to contain two additional attributes P^1 and P^2. For example, the node corresponding to w_2 has only one child node, so P^1 and P^2 of this node need to be calculated. The path from the root node to the node corresponding to w_2 is 35 $\|$ 20 $\|$ 15, so $P_2^1 = 35 \,\|\, 20 \,\|\, 15$ and $P_2^2 = f_{k'''}(P_2^1)$.

When a data user wants to search files containing w_7, he should calculate the trapdoor and send it to the server. In this example, the trapdoor for w_7 is 18. The black arrows indicate the path for searching 18. The server will return the file identifiers in A_7 and the proof P_7. When the data user wants to search files containing a specified keyword and the trapdoor of the keyword is 43, the red arrows indicate the path for searching for 43. Since the server did not find the node in MAVL and the node corresponding to w_1 is the last node on the search path, he will return P_1^1 and P_1^2. If the server honestly returns the search results, it can pass the verification. Otherwise, it will be found by the data user.

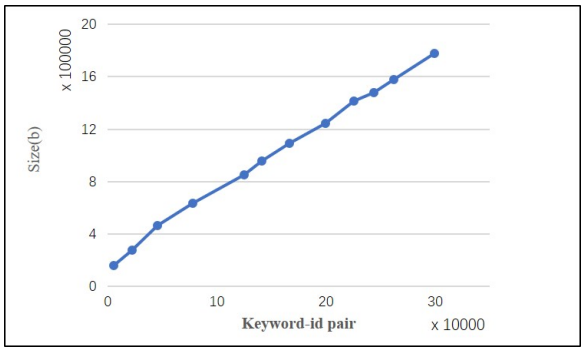
4. PERFORMANCE EVALUATION

To evaluate the performance of VSSE-AVL, we implement the VSSE-AVL in Windows 10 operation system using JavaScript language. And we measure the time and space overhead of the scheme and compare it with verifiable SSE-2 [11] of the same security. A series of experiments are conducted on simulation documents: the Railway Passenger Information Database. HMAC-MD5 is used to implement hash function. For each experiment, we perform it 10 times on a desktop equipped with an Intel(R) Core(TM) i5-9400F CPU (2.9GHz) and 16GB RAM, and take the average value as the result.

4.1. MAVL Construction



(a) The time cost of MAVL generation



(b) The size of MAVL

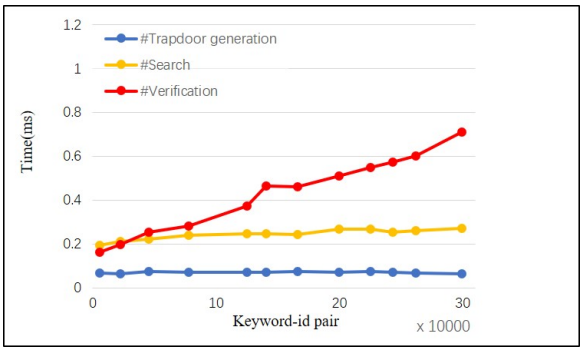
FIGURE 3. Index generation

Since MAVL is a modified AVL tree, the main calculations for constructing MAVL include calculating hash values, encrypting files, and building an AVL tree. In Fig.3(a), we measure the running time of MAVL generation phase. It can be seen that the index generation time is almost quadratic with the number of key-id pairs. Although the construction of MAVL is the most time-consuming phase, it is only executed once.

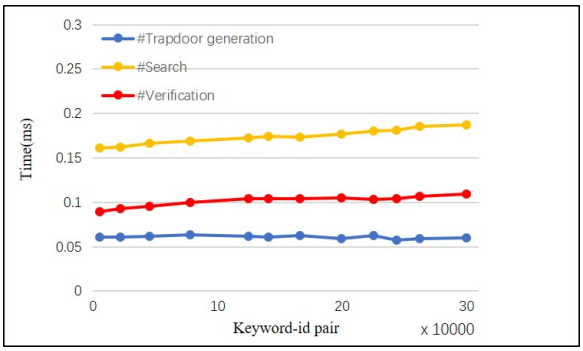
As shown in Fig.3(b), the size of MAVL has a roughly linear relationship with the number of keyword-id pairs. When the file set is 22.5M and the number of extracted keyword-id pairs is 243564, the MAVL's size is around 1.44M (not including the cipher text set). Compared with the file size, the index is almost less than one-sixteenth of it.

4.2. Search and Verification

We assume that the data owner has authorized search abilities for the data user, that is, the data owner sends the keys to the data user through a secure channel in advance. In this part, we do not consider the transmission time of keys. We analyze the experimental results based on the following two cases: the specified keyword exists in the database and the specified keyword does not exist in the database.



(a) Search results are not empty



(b) Search results are empty

FIGURE 4. Time cost

Fig.4(a) and Fig.4(b) correspond to these two cases respectively.

Once an authorized user wants to search files containing a specified keyword, the trapdoor must be generated first. The blue lines in Fig.4 illustrate the running time of trapdoor generation phase. As you can see, the trapdoor generation time is almost constant regardless of the number of keyword-id pairs. Because this phase only needs to calculate a hash function value.

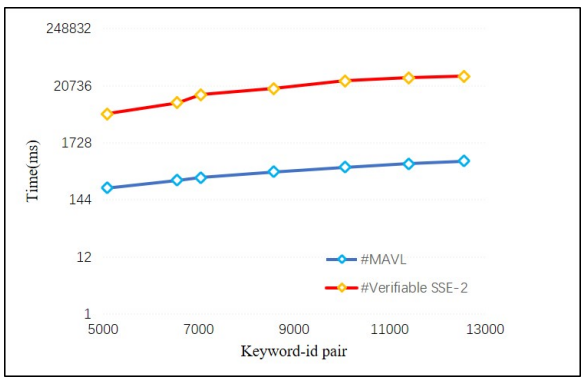
Upon receiving the trapdoor, the server searches over MAVL. The yellow lines in Fig.4 show the running time of keyword seach phase. An AVL tree is also a binary search tree, so MAVL should have sub-linear search complexity. As shown in Fig.4, there is roughly a logarithmic relationship between the keyword search time and the number of keyword-id pairs.

Whether the search results are empty or not, data users must verify their correctness and completeness. The red lines in Fig.4 show the running time of verification phase. As shown in Fig.4(a), the verification time is almost linear to the number of keyword-pairs. When the file set is 22.5M and the number of extracted keyword-id pairs is 243564, the time to search a specified keyword is about 0.25ms. When the search results are empty, it can be seen from Fig.4(b) that the verification time is about one-half of the keyword search time. And it increases slowly with the number of keyword-pairs.

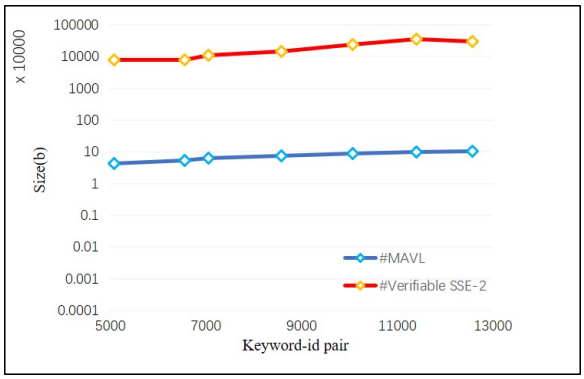
4.3. Comparison with verifiable SSE-2

Kurosawa et al. proposed verifiable SSE-2 based on SSE-2 in [7], but did not evaluate its performance. To be more equitable, we conduct experiments on the same file sets and keyword sets. In Fig.5, we show the time and space overhead of building index in verifiable SSE-2 and VSSE-AVL. Due to the huge difference in time and space overhead, the ordinate adopts a logarithmic scale. Fig.5(a) illustrates that the running time of the index generation phase of our scheme is about 3% of the verifiable SSE-2. Fig.5(b) illustrates that the index size of our scheme is about one thousandth of that of Verifiable SSE-2.

In Fig.6, we show the running time of trapdoor generation, keyword search, and verification phases of the two schemes. Due to their huge difference in time cost, we introduce the secondary axis (on the right) as the ordinate axis of the verifiable SSE-2. When the number of keyword-id pairs is 12569, the running time of verifiable SSE-2 is about 3.7 million times that of VSSE-AVL. Compared to verifiable SSE-2, the running time of our scheme is very short. And these experimental results show that our scheme is more efficient than verifiable SSE-2 in storage, search or verification.



(a) Index generation time



(b) Index size

FIGURE 5. Index generation

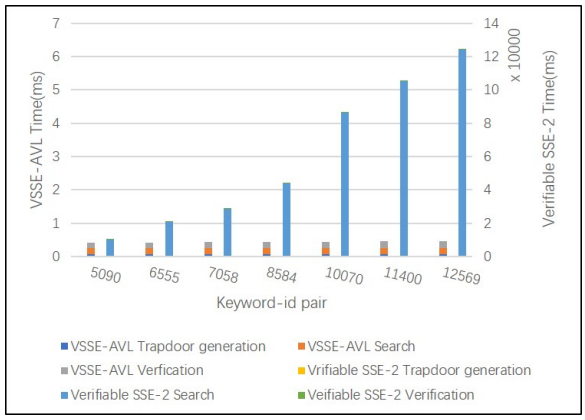


FIGURE 6. Time cost

5. CONCLUSION

In this paper, we proposed a verifiable symmetric search encryption scheme based on the AVL tree. The main idea is that MAVL is generated according to complete keyword labels. The number of nodes in the MAVL is the number of keywords (m), and its search complexity is $O(\log m)$. And VSSE-AVL could not only verify the correctness and completeness of the results even they are empty but also resist substitution attacks. A series of experiment results showed VSSE-AVL has the advantage of smaller storage capacity, shorter search time and shorter verification time.

DATA AVAILABILITY

The data underlying this article will be shared on reasonable request to the corresponding author.

FUNDING

This work is supported by the National Natural Science Foundation of China (No.62072276, No.61772311).

REFERENCES

[1] Kolodner, E. K. et al. (2011) A cloud environment for data-intensive storage services. *2011 IEEE third international conference on cloud computing technology and science*, Athens, Greece, 29 November-1 December, pp. 357–366. IEEE.

[2] Alba, A. et al. (2014) Efficient and agile storage management in software defined environments. *IBM Journal of Research and Development*, **58**, 5–1.

[3] Teiss, R. (2021). Data leakage attacks grew by 93 percent in 2020. Website. https://teissrecruitment.com/2021/01/28/data-leakage-attacks-grew-by-93-percent-in-2020/.

[4] Song, D. X., Wagner, D., and Perrig, A. (2000) Practical techniques for searches on encrypted data. *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, Berkeley, CA, USA, 14-17 May, pp. 44–55. IEEE.

[5] Goh, E.-J. (2003). Secure indexes. Cryptology ePrint Archive, Report 2003/216. https://ia.cr/2003/216.

[6] Chang, Y.-C. and Mitzenmacher, M. (2005) Privacy preserving keyword searches on remote encrypted data. *International conference on applied cryptography and network security*, New York, NY, USA, 7-10 June, pp. 442–455. Springer, Berlin Heidelberg.

[7] Curtmola, R., Garay, J., Kamara, S., and Ostrovsky, R. (2011) Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, **19**, 895–934.

[8] Li, J., Huang, Y., Wei, Y., Lv, S., Liu, Z., Dong, C., and Lou, W. (2019) Searchable symmetric encryption with forward search privacy. *IEEE Transactions on Dependable and Secure Computing*, **18**, 460–474.

[9] Song, Q., Liu, Z., Cao, J., Sun, K., Li, Q., and Wang, C. (2020) Sap-sse: Protecting search patterns and access patterns in searchable symmetric encryption. *IEEE Transactions on Information Forensics and Security*, **16**, 1795–1809.

[10] Chai, Q. and Gong, G. (2012) Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. *2012 IEEE International Conference on Communications (ICC)*, Ottawa, ON, Canada, 10-15 June, pp. 917–922. IEEE.

[11] Kurosawa, K. and Ohtaki, Y. (2012) Uc-secure searchable symmetric encryption. *International Conference on Financial Cryptography and Data Security*, Kralendijk, Bonaire, 27 February-2 March, pp. 285–298. Springer, Berlin Heidelberg.

[12] Hu, S., Cai, C., Wang, Q., Wang, C., Luo, X., and Ren, K. (2018) Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization. *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, Honolulu, HI, USA, 16-19 April, pp. 792–800. IEEE.

[13] Wang, J., Ma, H., Tang, Q., Li, J., Zhu, H., Ma, S., and Chen, X. (2013) Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. *Computer science and information systems*, **10**, 667–684.

[14] Sun, W., Wang, B., Cao, N., Li, M., Lou, W., Hou, Y. T., and Li, H. (2014) Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel and Distributed Systems*, **25**, 3025–3035.

[15] Zhu, J., Li, Q., Wang, C., Yuan, X., Wang, Q., and Ren, K. (2018) Enabling generic, verifiable, and secure data search in cloud services. *IEEE Transactions on Parallel and Distributed Systems*, **29**, 1721–1735.

[16] Kurosawa, K. and Ohtaki, Y. (2013) How to update documents verifiably in searchable symmetric encryption. *International Conference on Cryptology and Network Security*, Paraty, Brazil, 20-22 November, pp. 309–328. Springer, Berlin Heidelberg.

[17] Taketani, S. and Ogata, W. (2015) Improvement of uc secure searchable symmetric encryption scheme. *International Workshop on Security*, Nara, Japan, 26-28 August, pp. 135–152. Springer, Berlin Heidelberg.

[18] Zhu, X., Liu, Q., and Wang, G. (2015) Verifiable dynamic fuzzy search over encrypted data in cloud computing. *International Conference on Algorithms and Architectures for Parallel Processing*, Zhangjiajie, China, 18-20 November, pp. 655–666. Springer, Berlin Heidelberg.

[19] Liu, X., Yang, G., Mu, Y., and Deng, R. H. (2018) Multi-user verifiable searchable symmetric encryption for cloud storage. *IEEE Transactions on Dependable and Secure Computing*, **17**, 1322–1332.

[20] Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., and Lou, W. (2009) Enabling efficient fuzzy keyword search over encrypted data in cloud computing. *IACR Cryptol. ePrint Arch.*, **2009**, 593.

[21] Boneh, D., Di Crescenzo, G., Ostrovsky, R., and Persiano, G. (2004) Public key encryption with keyword search. *International conference on the theory and applications of cryptographic techniques*, Interlaken, Switzerland, 2-6 May.

[22] Abdalla, M. et al. (2005) Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *Annual international cryptology conference*, Santa Barbara, California, USA, 14-18, August, pp. 205–222. Springer, Berlin Heidelberg.

[23] Boneh, D. and Waters, B. (2007) Conjunctive, subset, and range queries on encrypted data. *Theory of cryptography conference*, Santa Barbara, California, USA, 21-24, February, pp. 535–554. Springer, Berlin Heidelberg.

[24] Wan, Z. and Deng, R. H. (2016) Vpsearch: achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data. *IEEE transactions on dependable and secure computing*, **15**, 1083–1095.

[25] Catalano, D. and Fiore, D. (2013) Practical homomorphic macs for arithmetic circuits. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Athens, Greece, 26-30, May, pp. 336–352. Springer, Berlin Heidelberg.

[26] Wang, J., Chen, X., Huang, X., You, I., and Xiang, Y. (2015) Verifiable auditing for outsourced database in cloud computing. *IEEE transactions on computers*, **64**, 3293–3303.

[27] Wang, J., Chen, X., Li, J., Zhao, J., and Shen, J. (2017) Towards achieving flexible and verifiable search for outsourced database in cloud computing. *Future Generation Computer Systems*, **67**, 266–275.

[28] Adelson-Velskij, G. M. and Landis, E. M. (1962) An algorithm for the organization of information. *Dokl. Akad. Nauk SSSR*, **146**, 263–266.

[29] Sedgewick, R. (2008) Left-leaning red-black trees. *Dagstuhl Workshop on Data Structures*.