An Efficient Two-Server Ranked Dynamic Searchable Encryption Scheme

YUXI LI\(^1\), (Student Member, IEEE), FUCAI ZHOU\(^2\), ZIFENG XU\(^2\), AND YUE GE\(^2\)

\(^1\)Key Laboratory of Intelligent Computing in Medical Image, Ministry of Education, Northeastern University, Shenyang 110000, China
\(^2\)Software College, Northeastern University, Shenyang 110000, China

Corresponding author: Yuxi Li (eliyuxi@gmail.com)

This work was supported by the Natural Science Foundation of China under Grant 61772127, Grant 61532007, and Grant 61472184.

ABSTRACT

Searchable encryption (SE) allows users to search over encrypted data without decrypting. In most existing SE schemes, server returns all matched files without relevance ranking, and the update mechanism are suffering with high communication and computation cost, which are not efficient enough to satisfy the real-life dynamic scenario. Addressing the above issues, we proposed TS-RDSE—a Two-Server Ranked Dynamic Searchable Encryption scheme. We integrate orthogonal vector and efficient homomorphic encryption cryptosystems to build a vector-level dynamic secure index, which simultaneously supports efficient dynamic update operations like deletion and insertion of files flexibly. Moreover, in order to rank the search results by relevance without decryption, we build a secure sorting protocol based on the widely-used tf-idf weighting formula and addition property of partial homomorphic encryption, which achieves accurate sorting for the search results while protecting the privacy of relevance scores. We give a comprehensive analysis of the correctness of TS-RDSE in the aspect of searching and sorting with mathematical proofs. The security analysis shows that TS-RDSE is secure against adaptive dynamic chosen-keyword attacks (CKA2) by honest-but-curious adversaries in random oracle model. The performance analysis shows that TS-RDSE has both a very light user workload and a moderate server workload, and it is superior to the existing approaches in terms of functionalities and expansibility. Extensive experiments on the real-world dataset validate our analysis and show that TS-RDSE is suitable for the real world cloud storage environment.

INDEX TERMS

Searchable encryption, dynamic update, homomorphic encryption, secure sorting.

I. MOTIVATION

With the rapid development of computer technology and the widespread of 5G service scenarios \([1]\), outsourced data and applications are growing rapidly. To save the cost of data management and local system maintenance, a large number of individual users and enterprises choose to migrate their data and business to the cloud and allow the cloud service provider to store and process their data by different demands. Since the data in the cloud is out of user’s physical control, user cannot effectively check the data’s security and consistency without totally trusting the cloud service provider. However, in recent years, there are more and more security incidents caused by security risks of cloud service providers \([2]\), \([3]\). Statistics show that about 92 percent of enterprises worry that outsourcing data to the cloud is a threat to data privacy \([4]\).

To avoid unauthorized access to user’s data from cloud administrators and hackers, user should encrypt the data at local before outsourcing. By encrypting sensitive data and storing it on untrusted cloud server, user can ensure that data confidentiality is guaranteed regardless of the privacy terms of cloud service provider. Since the traditional encryption scheme will change the original characteristics of data, it limits the server’s ability to perform search.

In order to let the server be able to compute search function as comprehensive as possible without decrypting the encrypted data, searchable encryption (SE) was born \([5]\), and quickly became a active research topic both in the theoretical researches (e.g., \([6]\)–\([8]\)) and in the practice (e.g., \([9]\), \([10]\)). The research object of SE is how to ensure the availability of outsourced encrypted data on the premise of protecting user privacy and data security. At present, there are still some shortcomings in the SE research, which are embodied in the following aspects:

Generally, the traditional SE schemes mainly consider the static setting. The forward index-based SE achieves directly
dynamic update, however, it suffers from linear (in the file number) search complexity. Many schemes use inverted index structure to speed up the search process in cloud server. When adding or deleting files, it is necessary to reconstruct the index structure which is inefficient in dynamic scenarios since the user’s workload and communication are overhead, and the leakage to the server is much. Although some schemes use additional data structures to perform update operations [6], [16], the access pattern and search pattern are leaked during update, and the high complexity of the dynamic updatable structure still leads to low update efficiency and allows only limited times update operations.

Moreover, most existing schemes do not capture the relevance between the search request and the retrieved files. Many ranked SE schemes [12]–[15] proposed to return all the files that match the search request to the user, who then rank the result in the local based on relevance weights. In large data scenarios, it is demand that the cloud server has the ability to sort the search results according to their correlation with search request, and place the most relevant files to user, to avoid transfer of irrelevant files and, hence, minimize the communication and computation cost at the user side. With ranked results, user can get the most relevance files more effectively. Many researchers do efforts to address how to sort search results in SE schemes to support ranked retrieval. However, the result sorting methods are either based on approximate ranking measurements [14]–[16] or need complex operations [11], [13], [18]. What’s more, many schemes did not take the keyword frequency into account, so the rank order of two files will be the same even if one contains all the searched keyword infrequently and the other one contains the same keyword very frequently. In terms of security, the proposed solutions are mainly based on higher assumptions [15], [19] (e.g., introducing trusted third parties) or specific security models [11], [18], [20]–[23] (e.g., the rank privacy are strictly weakened by allowing server obtain weight information).

Therefore, it is fair to say that how to design an efficient searchable encryption scheme in the two aspects of secure accurate result ranking and efficient dynamic update and secure accurate result ranking, is a very challenging task and still remains open. Addressing the above issues, we studied the state-of-art updatable search index structure and MPC-based comparison methods, and proposed TS-RDSE—a two-server ranked dynamic searchable encryption scheme that supports efficient updating and secure sorting. The contributions are summarized as follows:

- **We design a new dynamic secure index structure based on partial homomorphic encryption and orthogonal vector.** Benefiting from the property of the orthogonal vector, the special vector-level index simultaneously supports efficient search and dynamic update operations like deletion and insertion of files flexibly. The index includes the encrypted keyword-file relevance scores, which are computed by tf-idf weighting formula and encrypted by homomorphic encryption. To perform search, server only needs to conduct simple vector multiplication between search token and index to judge whether the search keywords appears, then compute the sum of the multi-keyword weight scores on the encrypted domain with addition property of partial homomorphic encryption.

- **We build a two-server secure sorting protocol to achieve ranked search on encrypted domain obliviously.** In this protocol, two non-collusive servers play the role of cloud service provider. By deploying the MPC-based comparison mechanism with Batcher’s sorting network, this protocol achieves efficient and accurate sorting while preventing the privacy of relevance score and the result order from the servers. Compared with the single server model, the two-server model reduces the communication cost between user and server, and also lower the information leaked during searching and sorting.

- **We give a comprehensive analysis of the correctness of TS-RDSE with mathematical proofs.** The formal analysis of security and performance shows that TS-RDSE has both a very light user workload and a moderate server workload while being secure against adaptive dynamic chosen-keyword attacks by honest-but curious adversaries in random oracle model. Extensive experiments on the real-world dataset validate our analysis and show that TS-RDSE is superior to the existing approaches in terms of functionalities and suitable for the real world cloud storage environment.

## II. RELATED WORKS

In recent literatures, many researchers proposed a number of searchable encryption schemes that focused on the two aspects of secure result ranking and dynamic update.

### A. DYNAMIC SEARCHABLE ENCRYPTION

Facing the requirement for dynamic scenario, researchers do much effort in studying the dynamic update method of secure index that can achieve both search and update operations. Among them, the schemes constructed by the forward index supports efficient updating of files and indexes [17], [18], which makes the dynamic update phase easy to develop applications in practice, such as Skyhigh Networks [9], CipherCloud [10] and Bitglass [24]. However, the forward index structure reveals a lot of plaintext-related information and the search efficiency is not very well. Kamara et al. proposed a dynamic searchable encryption scheme based on linked list structure in 2012 [6]. However, the update complexity of this scheme is linear in the number of files in the file collection which contain the updated keywords. They also extended their scheme to achieve parallel search [16] by deploying red-black tree in the index structure.

In 2014, Cash et al. [7] designed a dynamic SE scheme for big data scenario, which could support encrypted search for billions of records and keyword pairs. Fu et al. [25] designed a dynamic SE scheme that obtain semantic search based on keyword extension technique. However, it leaks
the information about the position of the newly added keyword as it is fixed. To reduce the update leakage, Bost constructed a dynamic searchable encryption scheme that satisfies forward security [8], but it only support searching and adding files without deleting. In 2018, Zuo et al. [21] built two dynamic searchable encryption schemes that realized forward security and partial backward security at the same time. However, it only allows limited times of update operations, and they did not provide a detailed privacy analysis. Later, Wang et al. [26] claimed that there are a few problems exist in the schemes in [21] which related to security and scalability, and they suggested a generic forward private DSSE with range queries, they also extended their scheme to achieve backward privacy by deploying another roundtrip between the user and the server. In the same year, Li et al. [27] proposed an integrity-verifiable multi-keyword dynamic searchable encryption scheme in response to the demand of multi-keyword searchable encryption in practical application scenarios. Etemad et al. [28] proposed to store the secure index in the user side and server side, which needs to be synchronized when updating, while the search key should be re-encrypted and updated after every search operation.

From the above schemes we can reach that the dynamic update mechanisms in SE would either cause extra information leakage when updating files, or needs high computation overload to build and update index structure which still leads to low update efficiency (even linear in the number of files containing the updated keywords) and allows only limited times update operations.

B. RANKED SEARCHABLE ENCRYPTION

Wang et al. [20] first proposed a ranked SE scheme, where the frequencies of keywords are encrypted with the file identifier, and the search time achieves sub-linear search time with the number of keywords based on the one-to-many order-preserving mapping. Cao et al. [11] extended the scheme in [20] and proposed a solution to support multi-keyword ranked encrypted search based on coordinate matching and inner product similarity. However, they build secure index based on Boolean representation, so files that contains the searched keywords have the same weight score.

Many schemes did not take the keyword frequency into account, so the rank order of two files will be the same even if one contains all the searched keyword infrequently and the other one contains the same keyword very frequently.

Term frequency-inverse document frequency (tf-idf) weighting formula is a common information retrieval method to evaluate the importance of a keyword to one document in a set of documents or a corpus [29]. Based on tf-idf, the follow-up works below were proposed to achieve better rank functionalities and greater efficiency. In 2013, Yu et al. [30] proposed a dynamic ranked SE scheme by separately storing tf values and idf values for each file in inverted index and auxiliary vectors. When updating, only the idf value of the updated keyword will be re-computed. Therefore, instead of updating all file vectors, only the auxiliary vectors are updated. In 2014, Strizhov et al. [14] combined tf-idf and inner product similarity (IPS) technology to design a new ranked SE scheme, where the weight scores are encrypted by ring-LWE-based variant of homomorphic cryptosystem. In 2015, Orenck and Savas [22] proposed a multi-keyword ranked searchable encryption scheme based on location sensitive hash (LSH) [32]. However, LSH is only suitable for similarity searches but cannot provide accurate ranking. Sun et al. [31] design a verifiable ranked searchable encryption scheme on tree structure by using cosine measure information retrieval technology in vectors space model. Their scheme can improve search efficiency at the expense of accuracy. Xia et al. [17] also proposed a ranking scheme where weights are encrypted using OPE scheme. Server can filter the retrieved identifier and send only top-k relevant results to user. The server clusters similar files based on k-nearest neighbors method, filters the retrieved identifier and send only top-k most relevant identifiers to user. In 2018, Shen et al. [13] proposed conjunctive encrypted query scheme based on the scheme in [7], they employ OPE to achieve result sorted. Unfortunately, OPE keeps the encrypted data sorted, which may cause leakage attack. The rank privacy are slightly weakened. In 2020, Guan et al. [23] designed a Cross-lingual multi-keyword rank search with semantic extension, which also speed up the sorting process by designing a top-k data retrieval protocol based on the heap binary tree structures.

C. TWO-SERVER SEARCHABLE ENCRYPTION

The designing of two-server based SE is a recent but active research topic both in the theoretical researches and in the practice. In 2014, Elmehdwi et al. [33] utilized two non-colluding clouds with Paillier cryptosystem to perform the encrypted k-nearest neighbors query, where the purpose of two-server setting is preventing the servers to learn the actual access patterns. Wang et al. [34] deployed two non-colluding servers jointly compute arithmetic functions over multi-key encrypted data. It reduced the privacy leakage by leveraging homomorphic encryption and proxy re-encryption. It contains many fairly complex processes like paring-based operations in elliptic curve during the two server interaction, which make the scalability slightly weakened. In 2015, Orenck et al. [22] proposed a two-server ranked SE scheme, where one performs search, and the other performs rank. But search operation requires more expensive asymmetric-based homomorphic encryption than [34]. In 2018, Rompay et al. [35] proposed a two-server multi-user SE scheme based on private set intersection (PSI), where the two servers receive either encrypted data or key. Meng et al. [14] proposed a relational database encryption scheme that support two-server range query with No-Random-Access (NRA) algorithm. In 2018, Shen et al. [13] extended the scheme in [7] to two-server setting so that conjunctive results are sorted by keyword occurrences encrypted using OPE. In 2019, Wu et al. [37]
also utilized secure classification over encrypted database in two cloud environments.

In practice, two non-colluding servers setting have been built and used in real-world applications for both performance and security reasons [18], [36], [38]. One server acts as a crypto cloud equipped with a secure processor, which stores the decryption key. The other one is the cloud service provider, which is typically provided by some large companies that have also commercial interests not to collude. Aiming at real-world application, Li et al. [18] designed an image retrieval service that supports similarity search and updates over encrypted images database with the prevalent model of two-servers in 2018, where the service provider just needs to perform the basic jobs of processing individual image. In 2019, Cheng et al. [38] constructed an efficient privacy-preserving auction mechanism for two-sided cloud markets which enables data-oblivious double auction by sharing keys with one market without the leakages of bidding privacy.

Following the works done above, we deploy our framework under two non-colluding honest-but-curious cloud servers to perform searching and sorting, one holding the encrypted data and the other the secret key.

II. BUILDING BLOCKS

A. PAILLIER CRYPTOGRAPHY

Paillier cryptosystem [39] is a probabilistic public key cryptographic system proposed by Pascal Paillier in 1999. It is a probabilistic asymmetric algorithm for public key cryptography. The message space \( M \) for the encryption is \( \mathbb{Z}_N \). \( N \) is a product of two large prime numbers \( p \) and \( q \). Define function \( L \) as \( L(x) = x - 1/n \). For a message \( m \in \mathbb{Z}_N \), we denote \([m] \in \mathbb{Z}_N^{+}\) to be the encryption of \( m \) with the public key \( pk \). In particular, Paillier encryption is three algorithms \( \mathcal{P} = \{ \text{P.KeyGen}, \text{P.Enc}, \text{P.Dec} \} \) as follows:

- \( \text{P.KeyGen}(1^k) \): To construct the public key, set an RSA modulus \( n = pq \) of \( k \) bits where \( p \) and \( q \) are large primes such that \( \text{gcd}(pq, (p-1)(q-1)) = 1 \). Let \( K = l\text{cm}(p-1)(q-1) \) and pick \( g \in \mathbb{Z}_N^{*} \). The public key is the pair \( pk = (n, g) \) and the secret is \( sk = K \).
- \( \text{P.Enc}_{sk, pk}(m) \): To encrypt a message \( m \in \mathbb{Z}_N \): choose \( r \in \mathbb{Z} \) and compute \( [m] = g^mr \mod n^2 \in \mathbb{Z}_N^{2} \).
- \( \text{P.Dec}_{sk, pk}(m) \): To decrypt a ciphertext \( c = [m] \) compute: \( m = (L([m]^k) \mod n^2) / (L(g^k) \mod n^2) \mod n \).

Since it is a secure and efficient cryptography along with the additive homomorphic property, it has been utilized in many existing applications such as E-voting. The addition homomorphic property is:

\[
\forall m_1, m_2 \in \mathbb{Z}_N, [m_1] \ast [m_2] = [m_1 + m_2]
\]

We also use the tool of a generalization of Paillier encryption [40]. The message space expands to \( \mathbb{Z}_N^{s+1} \) for \( s > 1 \), and the ciphertext space is under the group \( \mathbb{Z}_N^{s+1} \). This generalization allows one to doubly encrypt messages and use the additive homomorphism of the inner encryption layer under the same secret key. Note that the generalized version of Paillier uses \( g \in \mathbb{Z}_N^{s+1} \) as a public key while the secret is \( K' \) such that \( K' \mod n \in \mathbb{Z}_n \) and \( K' = 0 \mod K \), where \( K = lcm(p-1)(q-1) \) is the secret key of the original Paillier. We denote \([m] \) as ciphertext of \( m \) using the second layer. This extension allows a ciphertext of the first layer to be treated as a plaintext in the second layer. Moreover, this nested encryption preserves the structure over inner ciphertexts and allows one to manipulate it as follow:

\[
[[m_1]^{m_2}] = [[[m_1][m_2]]] = [[m_1 + m_2]]
\]

B. GOLDWASSER-MICALI CRYPTO SYSTEM

TS-RDSE also relies on Goldwasser-Micali (GM) cryptosystem [41]. The message space \( M \) for the encryption is \( \{0,1\} \). \( N \) is a product of two large prime numbers \( p \) and \( q \). For a message \( m \in \{0,1\} \), we denote \([m] \in \mathbb{Z}_N^2 \) to be the encryption of \( m \) with the public key \( pk \). Let \( J_p(x) \) be the Jocobi Symbol of \( x \) under \( p \). In particular, GM encryption is three algorithms \( GM = \{ \text{GM.KeyGen}, \text{GM.Enc}, \text{GM.Dec} \} \) as follows:

- \( \text{GM.KeyGen}(1^k) \): To construct the public key, set an RSA modulus \( n = pq \) of \( k \) bits where \( p \) and \( q \) are large primes such that \( \text{gcd}(pq, (p-1)(q-1)) = 1 \). Choose \( \delta \in \mathbb{Z}_N^* \), which satisfies \( (\delta, n) = 1 \). The public key is \( pk_{GM} = (\delta, n) \), and the private key is \( sk_{GM} = (p, q) \).
- \( \text{GM.Enc}_{sk, pk, GM}(m) \): To encrypt a message bit \( m \in \{0,1\} \), choose \( r \in \mathbb{Z}_N \), compute \([m] \) as follows:

\[
[m] = r^2 \delta^m \mod n.
\]

- \( \text{GM.Dec}_{sk, pk, GM}(m) \): To decrypt a ciphertext \( c = [m] \) using Jocobi Symbol to compute \( J_p(m) \) and \( J_q(m) \), if \( J_p(m) = J_q(m) = +1 \), then the message bit \( m \) is 0. Otherwise it’s 1.

It has the addition homomorphic properties:

\[
[m_1] \oplus [m_2] = [m_1 + m_2].
\]

C. TF-IDF WEIGHTING FORMULA

We use tf-idf formula [29] as weighting method of search results based on frequency of keywords in each document and the collection. Tf means term frequency—measures how frequently a term occurs in a document, while idf means inverse document frequency—measures how important a term is. Let \( N \) be the number of documents in the collection, \( p \) be the number of term \( t \) appears in a document \( d \), \( n \) be the number of terms in the document \( d \) and \( q \) be the number of documents with term \( t \) in it, then:

\[
tf(t, d) = p/n, \ \text{idf}(t, d) = \log(N/q).
\]

Document frequency of term \( t \) in document \( d \) is defined as:

\[
tf\text{-idf}(t, d) = tf(t, d) \cdot \text{idf}(t, d),
\]

If a document \( d \) does not contain \( t \) we set \( tf\text{-idf}(t, d) \) to zero.

IV. TS-RDSE MODEL

A. ARCHITECTURE

The architecture of TS-RDSE is shown in Figure 1. It consists of three types of entities: \( U \), \( S_1 \) and \( S_2 \).

- \( U \)—the owner of the outsourced file, can make keyword search request and obtain search result from the server.
To enable the assistant of files in the result by relevance, and sort them obliviously with the results, it can measure and evaluate the encrypted weight of the search request. Without revealing anything about the search time algorithms and protocols TS-RDSE consists of a tuple of six polynomial-time algorithms and protocols. Assume $\mathcal{U}$ has a file collection $\mathbf{f}$, TS-RDSE allows $\mathcal{U}$ to encrypt $\mathbf{f}$ to $\mathbf{c}$ at local and outsource the encrypted files $\mathbf{c}$ to $S_1$. To enable $S_1$ to search through $\mathbf{c}$, $\mathcal{U}$ builds a secure index $\gamma$, which contains a search index that is used for searching and a weight index that is used for ranking. $\mathcal{U}$ can later search requests to $S_1$ by keywords. $S_1$, upon receiving a search request, can perform searching in the secure index $\gamma$, and then retrieve all the matched file identities that satisfy the search request. Without revealing anything about the search result, it can measure and evaluate the encrypted weight of the files in the result by relevance, and sort them obliviously with the assistant of $S_2$. $S_1$ then filters the irrelevant results and returns the final ranked search results to $\mathcal{U}$. After receiving the results, $\mathcal{U}$ can decrypt them locally and get the ranked list of files. What’s more, user can also dynamically update the outsourced file collection and index on demand by sending update token to $S_1$.

**B. FORMAL DEFINITION**

**Definition 1:** TS-RDSE consists of a tuple of six polynomial-time algorithms and protocols $(\text{TS-RDSE}) = (\text{KeyGen}, \text{Setup}, \text{Token Gen}, \text{Search}, \text{Sort}, \text{Update})$ such that:

- $(K, SK, PK) \leftarrow \text{KeyGen}(1^k)$: is a probabilistic algorithm run by $\mathcal{U}$ that takes a security parameter $1^k$ as input, outputs user’s symmetric key $K$ and public/secret keys $PK, SK$.
- $(\gamma, c) \leftarrow \text{Setup}(K, PK, f)$: is a probabilistic algorithm run by $\mathcal{U}$ that takes the symmetric key $K$, public key $PK$ and a set of files $f$ as input, outputs a secure index $\gamma$ and a set of ciphertexts $c$.
- $\tau \leftarrow \text{TokenGen}(PK, K, W)$: is a deterministic algorithm run by $\mathcal{U}$ that takes symmetric key $K$, public key $PK$, and a set of keywords $W$ as input, outputs a search token $\tau$.
- $(S_1(R); S_2(\bot)) \leftarrow \text{Search}(S_1(\tau, \gamma); S_2(SK))$: is an interactive protocol run by $S_1$ and $S_2$ that $S_1$ takes search token $\tau$ and secure index $\gamma$ as input; $S_2$ takes secret key $SK$ as input, the output of $S_1$ is the search result $R$.
- $(S_1(R_1); S_2(SK)) \leftarrow \text{Sort}(S_1(R, PK); S_2(SK))$: is an interactive protocol run by $S_1$ and $S_2$ that $S_1$ takes the search results $R$ as input, and $S_2$ takes secret key $SK$ as input, the output of $S_1$ is a ranked search result $R_1$.
- $(S_1(\gamma', c')) \leftarrow \text{Update}(\mathcal{U}(K, PK, f); S_1(\gamma, c))$: is an interactive protocol running between $\mathcal{U}$ and $S_1$ ranked result. In the protocol, user takes keys $K, PK$, the file $f$ as input; $S_1$ takes the secure index $\gamma$ and encrypted file sets $c$ as input. After the protocol $S_1$ outputs the updated encrypted index $\gamma'$ and encrypted file set $c'$.

The correctness of TS-RDSE implies that, for all security parameters $k$, all keys $(K, SK, PK)$ generated by $\text{KeyGen}(1^k)$, for all $(\gamma, c)$ outputs by $\text{Setup}(K, PK, f)$, and for all sequences of search or update operations on $(\gamma, c)$, $\text{Sort}(S_1(\text{Search}(S_1(\text{TokenGen}(PK, K, W), \gamma); S_2(SK)), PK); S_2(SK))$ will always outputs the correct ranked results $R_1$, that is, not only satisfies the search request, but is sorted according to the corresponding weight scores.

**C. SECURITY DEFINITION**

The security definition for TS-RDSE is formalized by the four leakage functions $L_1, L_2, L_3$ and $L_4$, which denote the information leaked to $S_1$ and $S_2$ in different phases.
We formalize the ideal/real world paradigm for TS-RDSE. In the real world Real$(1^k)$, the protocols between the adversarial servers and user execute just like the real scheme. In the ideal world Ideal$(1^k)$, there exists two simulators Sim$_1$ and Sim$_2$ who can get the leakage information from leakage functions $L_1$ to $L_4$ and try to simulate the execution of $S_1$ and $S_2$ in the real world. We give the formal definition as follow.

Using the leakage, the simulators attempts to imitate the input and output behavior of the servers in the real instantiation. If Ideal$(1^k)$ and Real$(1^k)$ are indistinguishable by any PPT adversaries, then it means that the information more than the leakage, i.e. all data passed to the leakage functions that is not output by the leakage functions remains confidential.

The two paradigms are as follows.

**Definition 2:** Given the scheme described in Definition 1 and consider the following probabilistic paradigms where, $U$ is user, $A_1$ and $A_2$ are two non-colluding honest-but-curious PPT adversaries, Sim$_1$ and Sim$_2$ are two PPT simulators, and $L_1$ to $L_4$ as leakage functions:

**Real$(1^k)$:** It runs among the two adaptive PPT adversaries $A_1$, $A_2$ and the user $U$ using the real scheme. In this experiment, two sets of files $f_0$ and $f_1$ with the same number of files and keywords are send to the $U$. $U$ randomly chooses $b \leftarrow \{0, 1\}$ to generate a secure index $y_b$ of $f_b$ and sends $y_b$ to $A_1$. Subsequently, polynomial times queries $(q_1, \ldots, q_t)$ are made by user to $A_1$. For each query $q_i$, if the query is to search, then $A_1$ chooses two multi-keyword search requests $W_0(i), W_1(i)$ of equal length $|W_0(i)| = |W_1(i)|$ and send them to $U$. $U$ generates the search token $\tau_b(i)$ and send it to the adversary $A_1$, $A_2$ and $A_2$ executing $\text{Search}(A_1(y_b, \tau_b(i)); A_2)$ and get the ranked result $R_b(i)$. Subsequently, $A_1$ and $A_2$ execute $\text{Sort}(A_1(R_b(i), PK); A_2(SK))$ to get the ranked result $R_b(i)$. If the $i$-th query is to update, then $A_1$ chooses two files $f_0(i), f_1(i)$ of equal length $|f_0(i)| = |f_1(i)|$ and send them to $U$, $U$ generates the update token $\tau_b(i)$ and send it to the adversary $A_1$. Since $A_1$ and $A_2$ are adaptive adversaries, the results of each query can be used as input to generate the next query. At the end of all queries, the information $A_1$ received will be recorded as $\text{view}_A$. $A_2$ only participates in the interaction with $A_1$ and gets the interaction recorded as $\text{view}_A$. After the experiment, $A_1$ outputs $\text{Output}_A_1$, $A_2$ outputs $\text{Output}_A_2$.

**Ideal$(1^k)$:** It is run by the two simulators Sim$_1$ and Sim$_2$. Sim$_1$ and Sim$_2$ respond to queries using randomly generated data, with these leakage functions as the only input. By the information leaked from $L_1$ to $L_2$, Sim$_1$ builds simulated index structure $\tilde{\gamma}$. After that, there are $q$ polynomial times queries $(q_1, \ldots, q_t)$ send to Sim$_1$. For each query $q_i$, if the query is for searching, then Sim$_1$ chooses two multi-keyword search requests $W_0(i), W_1(i)$ of equal length $|W_0(i)| = |W_1(i)|$ and send them to $U$, $U$ generates the simulated search token $\tilde{\tau}_b(i)$ and send it to the adversary Sim$_1$. By the information leaked from $L_2$, Sim$_1$ and Sim$_2$ execute $\text{Search}(\text{Sim}_1(\tilde{\gamma}, \tilde{\tau}_b(i)); \text{Sim}_2(SK))$ and get the simulated search result $\tilde{R}_b(i)$. Subsequently, by the information leaked from $L_3$, Sim$_1$ and Sim$_2$ execute $\text{Sort}(\text{Sim}_1(\tilde{R}_b(i), PK); \text{Sim}_2(SK))$ to get the ranked result $R_0(i)$. If the $i$-th query is to update, then Sim$_1$ chooses two files $f_0(i), f_1(i)$ of equal length $|f_0(i)| = |f_1(i)|$ and send them to $U$, $U$ generates the update token $\tau_b(i)$ and send it to Sim$_1$. At the end of all queries, the information Sim$_1$ received will be recorded as $\text{view}_\text{Sim}_1$, Sim$_2$ only participate in the interaction with Sim$_1$ and get the interaction recorded as $\text{view}_\text{Sim}_2$. After the experiment, Sim$_1$ output $\text{Output}_\text{Sim}_1$, Sim$_2$ output $\text{Output}_\text{Sim}_2$.

We say that TS-RDSE is $(L_1, L_2, L_3, L_4)$-dynamic secure against adaptive chosen-keyword attacks secure in random oracle model if for all polynomial time $A_1$ and $A_2$, there exists polynomial time simulators Sim$_1$ and Sim$_2$ such that the following two distribution ensembles are computationally indistinguishable:

$$\text{Output}_{\text{Real}(1^k)} \approx \text{Output}_{\text{Ideal}(1^k)}.$$ 

**V. CONSTRUCTIONS**

**A. TECHNICAL OVERVIEW**

In TS-RDSE, before outsourcing files, $U$ encrypts his file collection using the symmetric encryption method which guarantees the CPA security of the original file. He also extracts keywords for each file, and builds an inverted index. The entries of the index correspond to the keyword, which leads to a set of files that contain this word. To support multi-keyword ranked search based on the relevance weight, $U$ should first measure the tf-idf scores for all the (keyword, file) pairs to evaluate the relevance, and insert the scores into a key-value weight index: The entries of the weight index correspond to the files IDs, which lead to a set of keywords as long as their weight scores.

We make use of two semi-secure homomorphic encryption schemes—Paillier and GM schemes to encrypt the weight index to a binary orthogonal vector-based secure weight index. Then, user publishes the public key and then outsources the encrypted data collection and the secure index to $S_1$.

To start a search, $U$ sends a search token to $S_1$. $S_1$ searches the matched files in the secure index. The homomorphic property of Paillier makes it possible for $S_1$ to compute the sum of weights for multiple keywords in a single file without decryption after the search process, so as to obtain the final encrypted weight for every search result. To rank the search results obviously, we design a two-server MPC-based secure sorting protocol—running between $S_1$ and $S_2$ to sort the search result. Our protocol relies on homomorphic properties of the Paillier cryptosystem to allow $S_1$ and $S_2$ to privately compare and swap pairs of ciphertexts. To achieve secure and efficient rank, we also require a parallel sorting network that performs comparisons in an oblivious way and guarantees that after performing a deterministic sequence of comparisons the result is sorted. We pick Batcher’s sorting [42] for our purposes.
Moreover, $\mathcal{U}$ can update the outsourced file set and secure index dynamically. The orthogonality of binary orthogonal vector-based index make it possible to support efficient and oblivious updates. Note that the search and update phases are both non-interactive with respect to $\mathcal{U}$. $\mathcal{U}$ needs only prepare his search/update token and send them to $S_1$.

### B. EXPLICIT CONSTRUCTION

Let $\text{SKE} = (\text{SKE}.\text{Gen}, \text{SKE}.\text{Enc}, \text{SKE}.\text{Dec})$ be a CPA-secure symmetric key encryption scheme. $\mathcal{F} : \{0,1\}^k \times \{0,1\}^k \rightarrow \{0,1\}^k$ be a pseudo-random function. Let $\mathcal{P} = (\mathcal{P}.\text{Gen}, \mathcal{P}.\text{Enc}, \mathcal{P}.\text{Dec})$ and $\mathcal{GM} = (\mathcal{GM}.\text{Gen}, \mathcal{GM}.\text{Enc}, \mathcal{GM}.\text{Dec})$ be Paillier encryption scheme and GM encryption scheme. The explicit construction is given as follows:

1) **KEY GENERATION**

To initialize the system, $\mathcal{U}$ needs to generate his own keys for encrypting files and building secure index. He first selects a $k$-bit string $k_F$ as the key of pseudo-random function $\mathcal{F}$, and performs the symmetric key generation algorithm $\text{SKE}.\text{Gen}$ to generate the symmetric secret key $k$. Moreover, he uses the Paillier encryption key generation algorithm $\mathcal{P}.\text{Gen}$ to generate the key pair $(pk, sk_p)$, and uses the GM encryption key generation algorithm $\mathcal{GM}.\text{Gen}$ to generate GM’s key pair $(pk_{GM}, sk_{GM})$.

To construct the secure index, $\mathcal{U}$ generates an orthogonal vector basis $V$: he first chooses $p$ Hadamard arrays $A_1, A_2, \ldots, A_p$, each of size is $e_1 \times e_i$ for $1 \leq i \leq p$ where every $e_i$ is either 2, 4 or 8. Later, he constructs $e_1 e_2 \ldots e_p$-sized matrix $A_M$ by the tensor product of these $p$ matrices given as $V \leftarrow A_M = A_1 \oplus A_2 \oplus \cdots \oplus A_p$. Then he chooses an orthogonal vector basis $(\tilde{v}_1, \ldots, \tilde{v}_n)$ from $V$, in which the length of each $\tilde{v}_i \in V$ is $1^k$.

Finally, $\mathcal{U}$ outputs $(K, SK, PK)$ as $\mathcal{U}$’s key, where $K = (k, k_F, V)$, $PK = (pk, pk_{GM})$, $SK = (sk_p, sk_{GM})$. $\mathcal{U}$ sends $PK$ to the two servers, sends $SK$ to server $S_2$ in secure channel.

**Algorithm 1 KeyGen**

**Require:**

Security parameters $1^k$

**Ensure:**

1. $K \leftarrow (\text{SKE}.\text{Gen}(1^k))$
2. $k_F \leftarrow \mathcal{R} \{0,1\}^k$
3. $(pk, sk_p) \leftarrow \mathcal{P}.\text{Gen}(1^k)$
4. $(pk_{GM}, sk_{GM}) \leftarrow \mathcal{GM}.\text{Gen}(1^k)$
5. Choose $p$ Hadamard arrays $A_1, A_2, \ldots, A_p$
6. $V \leftarrow A_M = A_1 \oplus A_2 \oplus \cdots \oplus A_p$
7. $(\tilde{v}_1, \ldots, \tilde{v}_n) \leftarrow \mathcal{R} V$
8. $K \leftarrow (k, k_F, V)$
9. $PK \leftarrow (pk, pk_{GM})$
10. $SK \leftarrow (sk_p, sk_{GM})$

2) **SETUP**

Let $f = (f_1, \ldots, f_{\ell_0})$ be the files set. For each file $f_i$, $\mathcal{U}$ extracts its keywords $w_{ji}$, encrypts each $f_i$ to $c_i$ using symmetric key encryption algorithm $\text{SKE}.\text{Enc}(c_i)$. Then $\mathcal{U}$ constructs the secure index $\gamma$, which consists of a search index $I$ and a weight index $S$.

- **Search index:** For each keyword $w_j$ in $w$, $\mathcal{U}$ first selects a vector $v_j$ from $V$, which acts as the secret vector of $w_j$. Then he initializes a binary index vector $v_i$ of length $\#f$: for each file $f_j$ in $f$, if $f_j$ contains keyword $w_i$, then set $v_i[j]$ to 1, otherwise set $v_i[j]$ to 0. Then $\mathcal{U}$ encrypts $w_i$ to $[w_i]_{\gamma_{ij}}$ by Paillier encryption algorithm $\mathcal{P}.\text{Enc}$ and encrypts $v_i[j]$ to $||v_i||$ by GM homomorphic encryption algorithm $\mathcal{GM}.\text{Enc}$. Let $||v_i||$ denote $\sum ||v_i||$. $\mathcal{U}$ generates the encrypted search index $I_{w_i}$ for each $w_i$ by calculating the vectors product: $I_{w_i} = \tilde{v}_i \cdot ||w_i|| \cdot [w_i]_{\gamma_{ij}}$. After all keywords are indexed, $\mathcal{U}$ selects a random number $r$ in $\mathcal{R}$ and a random vector $\tilde{v}_r$ in $V$ to build the final search index $I$:

$$I \leftarrow \begin{bmatrix} I_{w_1} \\ \vdots \\ I_{w_{\ell_0}} \end{bmatrix} \cdot I^{fw} + \tilde{v}_r \cdot r.$$  

- **Weight index:** $\mathcal{U}$ constructs a lookup table $S$ as the weight index. For each file $f_i$, $\mathcal{U}$ computes its unique pseudonym identifier $id_i$ by pseudo-random function $\mathcal{F}$ with his key $k: \mathcal{F}(k)$, and sets the entries of $S$ as the pseudonyms $id_i$. $S[id_i]$ points to a vector of length $\#w$ that contains every keyword that $f_i$ contains and the corresponding weight which are both in encrypted form.

In detail, for each keyword $w_j$ in $w_f$, $\mathcal{U}$ computes the weight score of $w_j$ in $f_j$: tf-idf$_{w_j}$ by tf-idf formula, and maps tf-idf$_{w_j}$ into $t_{w_j}^f$ in $\mathcal{R}$. Then $\mathcal{U}$ encrypts $t_{w_j}^f$ to $[t_{w_j}^f]_{\gamma_{ij}}$ by Paillier encryption algorithm $\mathcal{P}.\text{Enc}$, and calculates the vectors product $\tilde{v}_j \cdot [w_j]_{\gamma_{ij}} \cdot [t_{w_j}^f]_{\gamma_{ij}}$, where $\tilde{v}_j$ and $[w_j]_{\gamma_{ij}}$ have been generated when building the search index $I$. Finally, user selects a random number $r'$ in $\mathcal{R}$ and a random vector $\tilde{v}_r'$ in $V$, computes $S[id_i]$ as:

$$S[id_i] \leftarrow \sum_{j=1}^{\ell_0} (\tilde{v}_j \cdot [w_j]_{\gamma_{ij}} \cdot [t_{w_j}^f]) + \tilde{v}_r' \cdot r'.$$

Then user builds the final search index $S$ as follows:

$$S = \begin{bmatrix} S[id_1] \\ \vdots \\ S[id_{\ell_0}] \end{bmatrix} = \begin{bmatrix} [t_{w_1}^f] & \cdots & [t_{w_\ell_0}^f] \\ \vdots & \vdots & \vdots \\ [t_{w_1}^{fw}] & \cdots & [t_{w_{\ell_0}}^{fw}] \end{bmatrix} \cdot \begin{bmatrix} \tilde{v}_1[w_1] \\ \vdots \\ \tilde{v}_{\ell_0}[w_{\ell_0}] \end{bmatrix} + \begin{bmatrix} \tilde{v}_r' \cdot r' \\ \vdots \end{bmatrix}.$$
Subsequently, \( U \) integrates the search index and the weight into \( \gamma = (I, S) \), outputs \( \gamma \) as secure index and \( c = (c_1, \ldots, c_{\#}) \) as encrypted files, and sends them to \( S_1 \). The Setup description is shown in Algorithm 2.

### Algorithm 2 Setup

**Require:**

\( K, PK, f \)

**Ensure:**

\( \gamma, c \)

1. For \( 1 \leq i \leq \#f \) do
   
   \[ c_i \leftarrow SK\cdot E\cdot Enc_k(f_i) \]
2. \( c \leftarrow (c_1, \ldots, c_{\#}) \)
3. \( I \leftarrow \emptyset^{\#f} \)
4. For \( 1 \leq i \leq \#w \) do
5. \( w_i \leftarrow \emptyset^{\#f} \)
6. For \( 1 \leq j \leq \#f \) do
7. If \( w_j \in w_i \), then set \( 1 \leftarrow v_i[j] \)
8. For each \( v_i \) do return \( \emptyset \)
9. \( \|v_i]\| \leftarrow GM\cdot E\cdot Enc_{pk\cdot G,\cdot k}(v_i) \)
10. \( \|v_i]\| \leftarrow \sum_{w_i \in w_i} \|v_i]\| \)
11. \( r_{w_i} \leftarrow R_{Z_{n^2}} \cdot \gcd(r_{w_i}, n) = 1 \)
12. \( \|w_i\|_1 \leftarrow \mathcal{P}\cdot E\cdot Enc_{pk\cdot G,\cdot k}(w_i, r_{w_i}) \)
13. \( \tilde{v}_i \leftarrow R \cdot \mathcal{P}\cdot E\cdot Enc_{pk\cdot G,\cdot k}(w_i, r_{w_i}) \)
14. \( I_{w_i} \leftarrow \tilde{v}_i \cdot [\|v_i]\|]_{\|w_i]\|_1} \)
15. \( r' \leftarrow R \cdot \mathcal{P}\cdot E\cdot Enc_{pk\cdot G,\cdot k}(\tilde{v}_i, r'_{w_i}) \)
16. \( r_{w_i} \cdot r'_{w_i} \leftarrow R \cdot \mathcal{P}\cdot E\cdot Enc_{pk\cdot G,\cdot k}(v_i, r'_{w_i}) \)
17. \( I \leftarrow \sum_{i=1}^{\#f} (I_{w_i}) + \tilde{v}_i \cdot r' \)
18. \( S \leftarrow \emptyset^{\#f} \cdot \emptyset^{\#w} \)
19. For \( 1 \leq i \leq \#f \) do
20. \( id_i \leftarrow F_2(f_i) \)
21. For \( 1 \leq j \leq \#f \) do
22. \( t_{w_i} \leftarrow \mathcal{P}\cdot E\cdot Enc_{pk\cdot G,\cdot k}(v_i, f_i) \)
23. \( t_{w_i} \leftarrow R \cdot Z_{n^2} \cdot \gcd(t_{w_i}, n) = 1 \)
24. \( [t_{w_i}]_k \leftarrow \mathcal{P}\cdot E\cdot Enc_{pk\cdot G,\cdot k}(t_{w_i}, f_i) \)
25. \( r' \leftarrow R \cdot \mathcal{P}\cdot E\cdot Enc_{pk\cdot G,\cdot k}(\tilde{v}_i, r'_{w_i}) \)
26. \( S_{[id_i]} \leftarrow \sum_{j=1}^{\#f} (\tilde{v}_j \cdot [w_j]_{r_{w_j}} \cdot \|t_{w_i}\|) + \tilde{v}_i \cdot r' \)
27. \( \gamma \leftarrow (I, S) \)

### Algorithm 3 TokenGen

**Require:**

\( PK, K, \mathcal{W} \)

**Ensure:**

\( \tau \)

1. \( \mathcal{W} = \{w_1, \ldots, w_n\} \)
2. For \( 1 \leq i \leq \#\mathcal{W} \) do
3. \( w_i \leftarrow w_{i-1} \mod n \)
4. \( \tilde{r}_{w_i} \leftarrow r_{w_i} \mod n^2 \)
5. \( \left[\tilde{w}_i]\right]_{r_{w_i}} \leftarrow \mathcal{P}\cdot E\cdot Enc_{pk\cdot G,\cdot k}(\tilde{v}_i, \tilde{r}_{w_i}) \)
6. \( \tau_i \leftarrow \left[\tilde{w}_i]\right] \cdot \tilde{v}_i \)

4) SEARCH

After receiving a search token \( \tau = (\tau_1, \ldots, \tau_q) \), \( S_1 \) performs the multi-keyword encrypted search in the index \( \gamma \). It first computes the vector product \( R_{w_i} \) of each \( R_{\tilde{v}_i} \) with \( \gamma \) and computes the intersection of \( v_{w_i} \) to get the final binary vector \( v_{w_i} = v_1 \cup v_2 \cup \ldots \cup v_{\mathcal{W}} \). Then it returns \( v_{w_i} \) to \( S_1 \).

If \( v_{w_i} \) is 1, \( S_1 \) retrieves the vector in \( S_{[id_i]} \), and computes the vector product of \( S_{[id_i]} \) and each search token \( \tau_j \) to get the encrypted weight score for each keyword \( w_j \) in \( f_i \): \( [t_{w_i}^j] = S_{[id_i]} \cdot \tau_j \). Therefore, \( S_1 \) gets \( f_i \)'s encrypted weight score set \( ([t_{w_1}^i], \ldots, [t_{w_q}^i]) \) for the search keyword set \( \mathcal{W} \). Then \( S_1 \) can compute the product of them by Pailler encryption’s homomorphic addition property to get \( f_i \)’s aggregated weight score \( e_i \):

\[
e_i = \prod_{w_j \in \mathcal{W}} [t_{w_j}^i] = [t_{w_1}^i] \cdot [t_{w_2}^i] \ldots [t_{w_q}^i].
\]

Finally, \( S_1 \) generates the search results \( R = \{(id_1, e_1), \ldots, (id_d, e_d)\} \), where \( d \) is the number of matched files. The search phase is shown in Algorithm 4.

5) SORT

Since the weight scores \( (e_1, \ldots, e_d) \) in the search result \( R \) are encrypted by Pailler encryption, the randomness property prohibits \( S_1 \) from sorting the search result and returning the identifiers ranked by their weight scores to \( U \). To achieve ranking, we construct a secure sorting protocol running between \( S_1 \) and \( S_2 \) to privately sort the weight of the matched files. Our idea is to compare each pair \( (\|id_i\|, e_i), (\|id_j\|, e_j) \) in \( R \) based on the order \( \mathcal{P}\cdot Dec_{sk}(e_i) \) and \( \mathcal{P}\cdot Dec_{sk}(e_j) \) and then use the comparison as a blackbox building block for the sorting protocol. We pick Batcher’s sorting [39] for performing efficient parallel multi-time comparisons.
Algorithm 4 Search

Require:
\[ S_1(\tau, \gamma) ; S_2(\mathcal{L}) \]

Ensure:
\[ S_1(\mathcal{R}) ; S_2(\mathcal{L}) \]
1: \[ S_1(\mathcal{R}) \leftarrow 0^{|\mathcal{R}|} \]
2: For \( 1 \leq i \leq q \) do
3: \[ R_i \leftarrow (1 \cdot v_i) \mod n^2 \]
4: If \( R_i \neq 0 \), then \( \mathcal{R} \leftarrow \mathcal{R} \cup R_i \).
5: Else return \( \mathcal{R} \).
6: \[ S_1 \leftarrow S_2 : \mathcal{R} \]
7: \[ S_2(\mathcal{R}) \leftarrow 0^{|\mathcal{R}|} \]
8: For \( 1 \leq i \leq q \) do
9: \[ v_i \leftarrow \mathcal{P}.Dec_{sk}(R_i) \]
10: \[ v_W \leftarrow v_1 \cup v_2 \cup \ldots \cup v_q \]
11: \[ S_2 \leftarrow S_1 : v_W \]
12: \[ S_1(\mathcal{R}) \leftarrow 0^{|\mathcal{R}|} \]
13: For \( 1 \leq i \neq \# \) do
14: \[ v_W[i] \leftarrow 1, \text{ then} \]
15: For \( 1 \leq j \leq W \) do
16: \[ r_i \leftarrow \sum_{w \in W} r_{ij} \]
17: \[ R \leftarrow R \cup (\langle |d_i|, e_i \rangle) \]
18: Else return \( \mathcal{R} \).

To prevent \( S_2 \) from learning the order, \( S_1 \) generates the blind version \( R' \) of \( R = \{(|d_1|, e_1), \ldots, (|d_q|, e_q)\} \) by encrypting each pseudonym \( id_i \) of the matched files to \( |id_i| \). To sort \( R' \) of size \( d \) in a data-independent manner efficiently, Batcher’s sorting [31] is used to \( O((\log d)^2) \) data independent calls to \( \mathcal{SC} \). \( S_1 \) and \( S_2 \) first split \( R \) into two parts, sort them separately, then sort the odd elements and the even elements separately, and finally compare the elements indexed \( 2l \) and \( 2l + 1 \) for \( l = 1, 2, \ldots, (d/2) - 1 \) with 3, 4 with 5, and 6 with 7, respectively. We assume \( R' \) be the input, \( R' \) be an array which is the input of level \( i \).

For each level \( i \), for every pair \( (\langle |d_1|, e_1 \rangle, (|d_i|, e_i) \rangle) \), we construct a secure comparison protocol \( \mathcal{SC} \) for comparing each pair \( (\langle |d_1|, e_1 \rangle, (|d_i|, e_i) \rangle) \) in \( R \) based on the order \( \mathcal{P}.Dec_{sk}(e_i) \) and \( \mathcal{P}.Dec_{sk}(e_1) \). The goal of \( \mathcal{SC} \) is that \( S_1 \) obtains the encryption of the relation between the order of the plaintext of \( e_1 \) and \( e_i \) without learning neither the actual numbers nor the comparison result \( v \), where \( v = 1 \) if \( \mathcal{P}.Dec_{sk}(e_i) \leq \mathcal{P}.Dec_{sk}(e_1) \) and \( v = 0 \), otherwise. We also require \( S_2 \) to learn nothing about the relation between \( \mathcal{P}.Dec_{sk}(e_i) \) and \( \mathcal{P}.Dec_{sk}(e_1) \) but just assist \( S_1 \) to obtain an encryption of the comparison result. The main process that \( S_1 \) and \( S_2 \) perform in \( \mathcal{SC} \) is as follows:

\[ S_1(\tau, \gamma) ; S_2(\mathcal{L}) \]
1: \[ S_1(\tau, \gamma) \leftarrow 0^{\mathcal{L}} \]
2: For \( 1 \leq i \leq q \) do
3: \[ R_i \leftarrow (1 \cdot v_i) \mod n^2 \]
4: If \( R_i \neq 0 \), then \( \mathcal{R} \leftarrow \mathcal{R} \cup R_i \).
5: Else return \( \mathcal{R} \).
6: \[ S_1 \leftarrow S_2 : \mathcal{R} \]
7: \[ S_2(\mathcal{R}) \leftarrow 0^{|\mathcal{R}|} \]
8: For \( 1 \leq i \leq q \) do
9: \[ v_i \leftarrow \mathcal{P}.Dec_{sk}(R_i) \]
10: \[ v_W \leftarrow v_1 \cup v_2 \cup \ldots \cup v_q \]
11: \[ S_2 \leftarrow S_1 : v_W \]
12: \[ S_1(\mathcal{R}) \leftarrow 0^{|\mathcal{R}|} \]
13: For \( 1 \leq i \neq \# \) do
14: \[ v_W[i] \leftarrow 1, \text{ then} \]
15: For \( 1 \leq j \leq W \) do
16: \[ r_i \leftarrow \sum_{w \in W} r_{ij} \]
17: \[ R \leftarrow R \cup (\langle |d_i|, e_i \rangle) \]
18: Else return \( \mathcal{R} \).

Algorithm 5 Secure Comparison Protocol \( \mathcal{SC} \)

Require:
\[ S_1(\langle |d_1|, e_1 \rangle, (|d_i|, e_i), \mathcal{PK}); S_2(\mathcal{SK}) \]

Ensure:
\[ S_1(\langle |d_1|, e_1 \rangle, (|d_i|, e_i)), \mathcal{PK}); S_2(\mathcal{SK}) \]
1: \[ S_1(\langle |d_1|, e_1 \rangle, (|d_i|, e_i)), \mathcal{PK}); S_2(\mathcal{SK}) \]
2: \[ \mathcal{R} \leftarrow 0 \]
3: \[ \mathcal{L} \leftarrow 0 \]
4: \[ \mathcal{R} \leftarrow 0 \]
5: \[ \mathcal{L} \leftarrow 0 \]
6: \[ \mathcal{R} \leftarrow 0 \]
7: \[ \mathcal{L} \leftarrow 0 \]
8: \[ \mathcal{R} \leftarrow 0 \]
9: \[ \mathcal{L} \leftarrow 0 \]
10: \[ \mathcal{R} \leftarrow 0 \]
11: \[ \mathcal{L} \leftarrow 0 \]
12: \[ \mathcal{R} \leftarrow 0 \]
13: \[ \mathcal{L} \leftarrow 0 \]
14: \[ \mathcal{R} \leftarrow 0 \]
15: \[ \mathcal{L} \leftarrow 0 \]
16: \[ \mathcal{R} \leftarrow 0 \]
17: \[ \mathcal{L} \leftarrow 0 \]
18: \[ \mathcal{R} \leftarrow 0 \]
19: \[ \mathcal{L} \leftarrow 0 \]
20: \[ \mathcal{R} \leftarrow 0 \]
21: \[ \mathcal{L} \leftarrow 0 \]
22: \[ \mathcal{R} \leftarrow 0 \]
23: \[ \mathcal{L} \leftarrow 0 \]
24: \[ \mathcal{R} \leftarrow 0 \]
25: \[ \mathcal{L} \leftarrow 0 \]
26: \[ \mathcal{R} \leftarrow 0 \]
27: \[ \mathcal{L} \leftarrow 0 \]
28: \[ \mathcal{R} \leftarrow 0 \]
29: \[ \mathcal{L} \leftarrow 0 \]

After the sort phase, \( S_1 \) obtains the search result in a ranked array \( \mathcal{R}_y = \{(|d_1'|, e_1'), \ldots, (|d_q'|, e_q')\} \), which is a re-encryption of array \( \mathcal{R} \) using the first layer of Paillier that satisfies \( \mathcal{P}.Dec_{sk}(e_1') \leq \mathcal{P}.Dec_{sk}(e_2') \leq \ldots \leq \mathcal{P}.Dec_{sk}(e_d') \). Finally, \( S_1 \) sends the final ranked result \( \mathcal{R}_y \) to \( U, U \) can decrypt each \( |d_i'| \) in \( \mathcal{R}_y \) and get the ranked identities \( (id_1, \ldots, id_d) \). The sort phase is shown in Algorithm 6.
Algorithm 6 Sort

Require:

\[ S_1(R, PK); S_2(SK) \]

Ensure:

\[ S_1(R_\perp); S_2(\perp) \]

1: \[ S_1: \text{For } 1 \leq j \leq d \text{ do} \]
2: \[ [id_j] \leftarrow P.Enc_{pk}(id_j) \]
3: \[ \tilde{R} \leftarrow \{(id_1, e_1), \ldots, (id_d, e_d)\} \]
4: \[ R_1 \leftarrow R \]
5: \[ \text{For } 1 \leq j \leq O((logd)^2) \text{ do} \]
6: \[ R_{j+1} \leftarrow R_j \]
7: \[ \text{For } 1 \leq i \leq d \text{ do} \]
8: \[ (\{(id_j, e_j), (id_j', e_j')\} \leftarrow \text{pairs}_i.next \]
9: \[ (\{(id_j', e_j')\} \leftarrow SC \]
10: \[ (\{(id_j', e_j')\} \leftarrow SC \]
11: \[ R_s \leftarrow R_d \]

6) UPDATE

\( \mathcal{U} \) can dynamically update the outsourced file collection as well as the secure index. To add a new file \( f \), \( \mathcal{U} \) computes its pseudonym \( idf = F_k(f) \) and extracts its keyword set \( w_f = (w_{1f}, \ldots, w_{wf}) \). For every \( wi \) in \( w_f \), \( \mathcal{U} \) first updates its original index vector \( vi \) by setting \( vi'[f] \) as 1, generates \( wi \)'s new sub-index \( \tilde{I}_{wi} = v_i \cdot |v_i|^j \cdot [w_i]_{r_{wi}} \) computes tf-idf\( w_i \), and maps tf-idf\( w_i \) into the weight score \( r_{wi} \) in \( R \). Finally, \( \mathcal{U} \) randomly chooses \( r_{wi} \) from \( Z_n \), which satisfies \( gcd(r_{wi}, n) = 1 \), and computes \( f_{wi} = P.Enc_{pk}(r_{wi}, r_{wi}^n) \).

\( \mathcal{U} \) generates \( \gamma_{\phi}^f = (I_f, I'_f) \) which contains the new sub-index set \( I_f = (I_{w_1}, \ldots, I_{w_f}) \) and the original sub-index set \( I'_f = (I'_{w_1}, \ldots, I'_{w_f}) \). Then he randomly chooses \( r' \in R \), \( \tilde{v}_{r'} \in \mathcal{V} \), and computes \( \gamma_{\phi} = (id_j, \sum_{j=1}^{w_i \in w_f} (v_i \cdot [w_i]_{r_{wi}} \cdot [r_{wi}^n]_{r'}) + \tilde{v}_{r'} \cdot r' \). Afterwards, he sends the integrated update token \( \gamma_{\phi} = (\gamma_{\phi}^f, \gamma_{\phi}^r) \) to \( S_1 \).

\( S_1 \) updates the search index \( I \) to \( I' = \tilde{I} + I_{w_i} + I'_{w_i} \) based on the adding mechanism of orthogonal vector group, and updates the weight index \( S \) to \( S' \) by replacing the original vector that \( S[\gamma_{\phi}^f] \) pointed to with the new vector \( \gamma_{\phi}^r \).

\( S[\gamma_{\phi}^f] \) pointed to with the new vector \( \gamma_{\phi}^r \).

In Algorithm 7 we present our update protocol for adding new file. Deleting file is in a similar way. To avoid repetition, it’s not described here.

VI. ANALYSIS

A. CORRECTNESS

The correctness of TS-RDSE implies that, for all security parameters \( k \), all keys (\( K, SK, PK \)) generated by KeyGen\( (1^k) \), for all \( (\gamma, c) \) outputs by Setup\( (K, PK, f) \), and for all sequences of search or update operations on \( \gamma \), Sort\( (\gamma, \text{Sort}(\gamma)) \), \( \text{Search}(\gamma, \text{Search}(\gamma)) \), \( \text{Order}(\gamma, \text{Order}(\gamma)) \), \( \text{Update}(\gamma, \text{Update}(\gamma)) \), \( \text{Delete}(\gamma, \text{Delete}(\gamma)) \) will always output the correct ranked results \( R_\gamma \), that is, not only satisfies the search request, but is sorted according to the corresponding weight scores. We mainly divided the correctness of TS-RDSE in the following two parts:

1) CorrectSearch:

The correctness of search implies that the search algorithm \( \text{Search} \) always output the correct results.

Proof: To search for keywords \( W = \{w_1, \ldots, w_q\} \), \( \mathcal{U} \) computes \( \tilde{w}_i \) and \( r_{wi} \) that satisfies \( \tilde{w}_i \cdot w_i \equiv 1 \mod n \) and \( r_{wi} \cdot r_{wi} \equiv 1 \mod n^2 \). Then he generates \( [\tilde{w}_i]_{r_{wi}} = P.Enc_{pk}(\tilde{w}_i, r_{wi}) \), \( g^w_i \cdot \tilde{r}_{wi} \cdot \tilde{v}_1 \), and \( g^w_i \cdot \tilde{r}_{wi} \cdot \tilde{v}_1 \). Upon receiving the search token \( \tau = (\tau_1, \ldots, \tau_q) \), \( \mathcal{U} \) computes the update token \( \tau_i = [\tilde{w}_i]_{r_{wi}} \cdot \tilde{v}_1 \). If \( w_i \) does exist in the keyword set \( w \), it can be deduced that the sub-index \( I_{w_i} \) is included in the search index \( I \), so \( I_{w_i} \) can be represented as \( I = \sum_{j=1}^{w_i} (I_{w_i}) + \tilde{v}_1 \cdot r \). So \( S_1 \) can
compute the search result $R_{w_i}$ by the following equation:

$$ R_{w_i} = (I \cdot \tau_{i}) \mod n^2 $$

$$ = (\sum_{j=1}^{w} (I_{w_j} + v_r \cdot r) \cdot [\tilde{v}_i]_{r_{w_j}} \cdot \tilde{v}_i) \mod n^2 $$

$$ = (\sum_{j \neq w} (I_{w_j} + \tilde{v}_i \cdot v_r \cdot r) \cdot g^{\tilde{v}_i h_{w_j}} \cdot \tilde{v}_i) \mod n^2 $$

$$ = (\sum_{j \neq w} (\tilde{v}_j \cdot [v_{j}] \cdot g^{w_j h_{w_j}}) + \tilde{v}_i \cdot [v_{i}] \cdot g^{w_i h_{w_i}} \cdot v_r \cdot r) \cdot g^{\tilde{v}_i h_{w_j}} $$

$$ \cdot \tilde{v}_i \mod n^2 $$

$$ = (\sum_{j \neq w} (\tilde{v}_j \cdot [v_{j}] \cdot g^{w_j h_{w_j}}) + \tilde{v}_i \cdot [v_{i}] \cdot g^{w_i h_{w_i}} \cdot v_r \cdot r) \cdot g^{\tilde{v}_i h_{w_j}} \cdot \tilde{v}_i \mod n^2 $$

$$ = [v_{i}] \cdot g^{w_i h_{w_i}} \cdot g^{\tilde{v}_i h_{w_j}} $$

$$ = [v_{i}]. $$

According to the property of orthogonal vector basis, if the set of orthogonal vectors $\mathcal{V}$ contains $\tilde{v}_i$ and $\tilde{v}_j$, where $\tilde{v}_i \neq \tilde{v}_j$, then $\tilde{v}_i \cdot \tilde{v}_j = 0 \mod n^2$; if $\tilde{v}_i = \tilde{v}_j$, then $\tilde{v}_i \cdot \tilde{v}_j = 1 \mod n^2$. Because $w_{i-w_{j}} \equiv 1 \mod n$ and $\tilde{r}_{w_{i-w_{j}}} \equiv 1 \mod n^2$, $S_1$ can get the result $R_{w_i} = [v_{i}] \cdot g^{w_i h_{w_i}} \cdot g^{\tilde{v}_i h_{w_j}} \mod n^2 = [v_{i}] \mod n^2$, which is GM ciphertext of the corresponding binary index string of $w_i$.

If the keyword $w_i$ do not exist in the keyword set $w$, $S_1$ can compute the search result $R_{w_i}$ by the following equation.

$$ R_{w_i} = (I \cdot \tau_{i}) \mod n^2 $$

$$ = (\sum_{j=1}^{w} (I_{w_j} + v_r \cdot r) \cdot [\tilde{v}_i]_{r_{w_j}} \cdot \tilde{v}_i) \mod n^2 $$

$$ = (\sum_{j=1}^{w} (\tilde{v}_j \cdot [v_{j}] \cdot g^{w_j h_{w_j}}) + \tilde{v}_i \cdot [v_{i}] \cdot g^{w_i h_{w_i}} \cdot v_r \cdot r) \cdot [\tilde{v}_i]_{r_{w_j}} \cdot \tilde{v}_i \mod n^2 $$

$$ = 0. $$

2) Correct$_{SC}$

The comparison correctness implies that the secure comparison protocol $SC$ for the encrypted weights $e_x$ and $e_y$ from $\langle ([id_{i}], e_x), ([id_{j}], e_y) \rangle$ in each round of sorting is correct.

**Proof:** Suppose $D_a$ and $D_b$ are two integers with length $l$—bits, which represent the plaintexts of $e_x$ and $e_y$ separately. The value $z = D_a - D_b + 2^l$ is an integer with length $(l + 1)$—bits. We can see that if the $l$—th bit of $z$ is 1, then $D_a \leq D_b$; if the $l$—th bit of $z$ is 0, then $D_a \geq D_b$. The $l$—th bit of $z$ can be represented as $z \div 2^l$. So $z = 2^l(z \div 2^l) + (z \mod 2^l)$, where $0 \leq (z \mod 2^l) \leq 2^l$. Based on the above observation, we let $S_1$ compute $[z]$ as $2^l[e_x e_y]^{-1}$ mod $n^2(l > k)$, and blind it by computing $[d] := [z][r] \mod n^2$. Because the value of $z$ is blinded by a randomness $r$ to a new value $d: d = z + r$, so $d = 2^l(d \div 2^l) + (d \mod 2^l) = 2^l(z \div 2^l) + (r \div 2^l) + ((z \mod 2^l) + (r \mod 2^l))$.

We let $d_l$ be the $l$—th bit of $d$, $r_l$ be the $l$—th bit of $r$, and $\lambda$ be the comparison result of $d_l$ and $r_l$. If $d_l < r_l$, then $\lambda = 1$. We can compute $||v||$ by $||d|| ||r|| ||\lambda||$. For operations over bits using GM encryption, we don’t have this problem as we are operating on $F_2$. Due to the homomorphic property of GM encryption, we can get $||d_l|| ||r_l|| ||\lambda|| = ||d_l \oplus r_l \oplus \lambda||$. The correctness will hold as long as we do not experience carry-overs modulo $N$. In particular, this implies that $l + 1 + \lambda \leq \log_2 N$. For operations over bits using GM encryption, we don’t have this problem as we are operating on $F_2$. Due to the homomorphic property of GM encryption, we can get $||d_l|| ||r_l|| ||\lambda|| = ||d_l \oplus r_l \oplus \lambda||$. Therefore, the correctness of computing the $l$—th bit of $z$ can be translated into the correctness of computing $d_l \oplus r_l \oplus \lambda$.

If $(z \mod 2^l) + (r \mod 2^l) < 2^l$, then $d \mod 2^l = z + r \mod 2^l$; if $(z \mod 2^l) + (r \mod 2^l) \geq 2^l$, then $d \mod 2^l = z + r \mod 2^l + 2^l$. The above equations can be represented as $d = z + r + 2^l \lambda$, where $\lambda = 0 \Rightarrow (z \mod 2^l) + (r \mod 2^l) < 2^l$. If $\lambda = 0$, then $d \mod 2^l = (z \mod 2^l) + (r \mod 2^l)$; if $\lambda = 0$ and $d \mod 2^l = (z \mod 2^l) + (r \mod 2^l) > 2^l$, then $d \mod 2^l = (z \mod 2^l) + (r \mod 2^l) + 2^l \lambda$. So $\lambda = 0 \Rightarrow d \mod 2^l = (z \mod 2^l) + (r \mod 2^l) + 2^l \lambda$. On the other hand, if $d \mod 2^l = (z \mod 2^l) + (r \mod 2^l) + 2^l \lambda$, then $d \mod 2^l = z + r + 2^l \lambda$. Since $(z \mod 2^l) + (r \mod 2^l) < 2^l$, we can get $d \mod 2^l > 2^l \lambda = d_l > r_l$, therefore $\lambda = 0$.

Therefore, since the value of $z \mod 2^l$ is 0 or 1, the calculation of $z \mod 2^l = (d \mod 2^l) - (r \mod 2^l) \lambda$ mod 2 = $d_l - r_l - \lambda$ mod 2 = $d_l \oplus r_l \oplus \lambda$ mod 2. Therefore, the plaintext of the $l$—th bit of $z$ can be measured as $d_l \oplus r_l \oplus \lambda$; and the GM ciphertext of the $l$—th bit of $z$ can be measured as $||d|| ||r|| ||\lambda||$. In conclusion, the process of comparison of $e_x$ and $e_y$ from $\langle ([id_{i}], e_x), ([id_{j}], e_y) \rangle$ is correct.

3) Correct$_{Sort}$

The sort correctness implies that sorting the key-value array $R = \langle ([id_{1}], e_1), \ldots, ([id_{2}], e_2) \rangle$ according to the weight scores is correct, which means the sort result $R_s = \langle ([id_{1}], e_{1'}), \ldots, ([id_{j}], e_{j'}) \rangle$ is a re-encryption of array $R$ that satisfies $\mathcal{P}.Dec_{sk_y}(e_{1'}) \leq \mathcal{P}.Dec_{sk_y}(e_{j'}) \leq \mathcal{P}.Dec_{sk_y}(e_{d'})$.

**Proof:** The proof mainly focus on that, for any search result $R_s = \langle ([id_{1}], e_{1}), \ldots, ([id_{d}], e_{d}) \rangle$ of length $d$, first sorting separately the first and second halves, then sorting separately the odd-indexed keys and the even-indexed keys, and finally comparing-and-switching the keys indexed 2$l$ and 2$l + 1$ for $l = 1, 2, \ldots, (d/2) - 1$ results in a sorted array $R_s = \langle ([id_{1}], e_{1}), \ldots, ([id_{j}], e_{j'}) \rangle$, which satisfies $\mathcal{P}.Dec_{sk_y}(e_{1'}) \leq \mathcal{P}.Dec_{sk_y}(e_{j'}) \leq \mathcal{P}.Dec_{sk_y}(e_{d'})$.

After the two halves of $R_s$ have been sorted separately, it is obvious that for all $q$ between 1 and $d$ except for 1 and $n/2 + 1$, the $(q - 1)$—th element of the array is less than the $q$—th. Call the $(q - 1)$—th key the predecessor of the $q$—th. Note that 1 and $n/2 + 1$ are both odd. Every even-indexed score has as its predecessor a number smaller than itself (since any even-indexed score and its predecessor are in the same half),
so the $l$-th smallest even-indexed score must be larger than at least $l$ odd-indexed scores. Similarly, every odd-indexed key (with the possible exception of two scores, namely the 1st and $((n/2)+1)$-th) has as its predecessor a number smaller than itself, so the $(l+1)$-th smallest odd-indexed score must be larger than at least $l-1$ even-indexed scores.

We denote by $D_q$ the $q$-th indexed score after sorting the first and second halves and the even and odd scores, which satisfies $D_q = P.D Ecc_{Sk}(e_q)$. We have just argued that $D_{2l-1} \leq D_{2l}$ and $D_{2l-2} \leq D_{2l+1}$, for any appropriate $l$. Since we have sorted the even indexed scores and the odd indexed scores, we also know that $D_{2l-1} \leq D_{2l}$ and $D_{2l-2} \leq D_{2l+1}$. Thus, if we group the elements in pairs $(D_{2l}, D_{2l+1})$ for each appropriate $l$, we see that both elements of a pair are greater or equal to both elements of the previous pair. To finish the sort after sorting the odds and evens, it is therefore only necessary to compare the $(l+1)$-st smallest even-indexed key $D_{2l+1}$ to the $l$-th smallest even-indexed key $D_{2l}$ for each appropriate $l$ to determine its rank completely; this is precisely what the final step does. So we can get that $D_1 \leq D_2 \leq \ldots \leq D_d$.

Therefore, the output of a correct ranker result is $\sigma_1 = \{[[id_1], e_1], \ldots, [[id_d], e_d]\}$.

### B. Security Analysis

First, we analyze the information leaked to the servers $S_1$ and $S_2$ in TS-RDSE. The formal definition will be given afterwards. Our goal is to prove that, any PPT adversary can obtain no information about the data and queries, except the information in the leakage functions. The leakage functions are described as follows:

- In the setup phase, given the encrypted secure index $\gamma = (I, S)$ and the ciphertexts $c$, $S_1$ can get the size of the search index vector $I$, the number of elements $\#S[id]$ of $S$, the length of each set $S[id]$ and the size of each ciphertext $c \in c$. We denote these by $L_1$, i.e.,

$$L_1 = ([I], \#S[id], [S[id]], |w|, |c|, \varepsilon)$$

- In the search phase, each search query, the search operation reveals to $S_1$ the association between each keyword $w$ and the pseudonym of the file containing the keyword. Specifically, let $Q = (\tau_1, \ldots, \tau_q)$ be the set of search tokens, where $l_{max}$ be the longest search token in the set $Q$, and the search history can be expressed as a binary fourth-order matrix $Q$ with the size of $n_q \times n_q \times l_{max} \times l_{max}$. Moreover, during the interaction with $S_1$, $S_2$ get $R_i$ for each query. Therefore, after $q$ times queries, the search operation reveals to $S_2$ a binary two-order matrix $R_{q \times q}$. We denote these by $L_2$, i.e.,

$$L_2 = ([([id(f)]_{f \in f}, id(w), Q)_{w \in w}], (R_{q \times q}))$$

- In the sort phase, $S_1$ and $S_2$ need to interact with each other. In each round of sorting interaction, the knowledge that $S_1$ can learn the information obtained from running $SC: (e_x, e_l, I)$; where $S_2$ can learn $(||, \lambda)$. After multiple rounds of interaction, $S_2$ also get the number of rounds of interaction—$(logd)^2$, then he can infer the number of files $d$ included in the results. We denote these by $L_3$, i.e.,

$$L_3 = ((\sum_{i=1}^{(logd)^2} e_{x,i}, e_{y,i}, I))_{S_1}, ((\sum_{i=1}^{(logd)^2} (||z_i, \lambda_i)), d)_{S_2})$$

- In the update phase, from the update token $\tau_f = (\tau_1, \tau_2)$, $S_1$ can learn the ciphertext size of update file $c_f$, get the index set size $\#f$ corresponding to the pseudonym of the update file $id_f$, so as to know the number of keywords $\#w_f$ it contains and the length $[S[id]]$ of the set $S[id]$ in the weight index $S$. We denote these by $L_4$, i.e.,

$$L_4 = ([c_f], \#f, \#w_f, [S[id]])$$

Now we use the following theorem to claim that the construction of TS-RDSE is $(L_1, L_2, L_3, L_4)$-secure against adaptive chosen-keyword attacks secure in random oracle model.

**Theorem 1:** If the $SK\Sigma$ scheme, Paillier and GM encryption cryptosystem are CPA-secure, $F$ is pseudo-random function, and the DKG protocol is proved semantic secure in random oracle model then the construction of TS-RDSE is secure against adaptive chosen-keyword attacks in the random oracle model.

**Proof:** The primary goal of providing this proof is to construct two simulators $Sim_1, Sim_2$ that can generate the simulated values in $Ideal(1^k)$ using the information given in these leakage functions described above.

We show that the outputs of the simulator $Sim_1$ and $Sim_2$ are indistinguishable from the view of $A_1$ and $A_2$ in $Real(1^k)$ using a sequence of hybrid games, where each of them simulates one or more non-revealed encrypted values than the previous game. As a consequence the first hybrid game corresponds to $Real(1^k)$ and the last one corresponds to $Ideal(1^k)$.

Given the information received from $L_1$, the simulator $Sim_1$ could learn the length and the structure of encrypted index $I$, the length of the set $S[id]$ in the weight index $S$ and the length of each ciphertext $c \in c$. Then it can use randomly chosen strings to construct these structures and produce these values as the simulated one $(I, S, \tilde{c})$. For all $f \in f$, computes $\tilde{c}_f = SK\Sigma.Enc_{Kf}(0/1)$. The simulated ciphertext $\tilde{c}$ are all generated by $(\tilde{c}_1, \ldots, \tilde{c}_n)$, so the CPA security of the $SK\Sigma$ encryption scheme prevent the adversary to distinguish $c$ from $\tilde{c}$. Pading random vector $\tilde{v}_r \cdot r$ and the pseudo-randomness of the ciphertext of Paillier and GM schemes in the indexes $(I, S)$ prevent the adversary to distinguish $(I, S)$ from $(I, \tilde{S})$. Therefore, $Output_{Real}(I, S, c) \approx Output_{Ideal}(I, S, c)$.

Given the information received from $L_2$, the simulator $Sim_1$ should respond the simulated search token during the adversary’s queries. These steps become more complex due to the fact that simulator needs to track the dependencies between the information revealed by these queries to ensure
consistency among these simulated tokens. From the interact, the simulator \( Sim_1 \) learn the search result \( R \) and the binary string \( y_{1W} \) satisfying the search request by multiplying \( t_1 \) and the search index vector \( \tau_i \). It can locate the pseudonym set of the corresponding file set \( D = (id_1, \ldots, id_d) \) through the information in \( y_{1W} \), and obtain the corresponding correlation encrypted weight \((e_1, \ldots, e_d)\). In the process mentioned above, the association between each keyword identifier \( id(w) \) and the pseudonym of the file containing the keyword. Specifically, let \( Q = \tau_1, \ldots, \tau_d \) be the set of search tokens, \( l_{max} \) represents the longest search token in the set \( Q \), and the search history can be expressed as a binary fourth-order matrix \( Q_{ny\times n_q\times l_{max}\times l_{max}} \). Therefore, Output\(^{\text{Real}}\)\(_{A_1}(I, S, c) \approx \) Output\(^{\text{Ideal}}\)\(_{A_1}(I, S, c, e) \).

Given the information received from \( L_3 \), the simulator \( Sim_2 \) could learn the information obtained from the leakage in the SC protocol \((\|z\|, [\lambda])\) and the number of rounds of interaction \((\log d)^2\). The knowledge that \( Sim_1 \) can learn in each round includes \((e_1, e_2, \ldots, e_d)\). Since \( A_1 \) and \( A_2 \) interact with each other for comparing values by \((\log d)^2\) times in real game, the simulators \( Sim_1 \) and \( Sim_2 \) should simulate \( A_1 \) and \( A_2 \) with the leakage functions \( L_3 \) by each pairs \((\log d)^2\) times in Batché’s protocol to get the final simulation value.

At every pair, \( A_1 \)'s view can be denoted as view\(_{A_1} = (e_1, e_2, l, pk_{GM}, pk_{P}, r; [\|\lambda\|], [z])\), in which \( pk_{P} \) is the secret key of Paillier Encryption. Given \((e_1, e_2, l, pk_{GM}, pk_{P})\), we can build \( Sim_1 \) to simulate \( A_1 \):

- choose \( R_S \leftarrow (0, 2^{λ+i}) \cap Z \)
- randomly choose \( λ, ˜z, \) generate \([\|\lambda\|], [\|\tilde{z}\|]\)
- output view\(_{Sim_1} = (e_1, e_2, l, pk_{GM}, pk_{P}, R_S; [\tilde{z}])\).

For view\(_{A_1}\) and view\(_{Sim_1}\), \( r \) is extracted from uniform distribution \((0, 2^{λ+i}) \cap Z\), and \([\|\tilde{z}\|]\) is the second layer ciphertext of Paillier encryption, which is randomness. So \((e_1, e_2, l, pk_{GM}, pk_{P}) = (e_1, e_2, l, pk_{GM}, pk_{P}, r, [\tilde{z}])\).

Besides, \( r \) and \( R_S \) are extracted from the same uniform distribution, based on the CPA-secure Paillier encryption, we can get: \( A_1 \)'s view and \( Sim_1 \)'s view are computationally indistinguishable.

In a similar way, at every pairs, \( A_2 \)'s view can be denoted as view\(_{A_2} = (sk_{GM}, sk_{P}, [\|\|], [\|\lambda\|])\). We can build \( Sim_2 \) to simulate \( A_2 \):

- randomly choose \( \tilde{\lambda}, \) generate \([\|\tilde{\lambda}\|]\), to represent that \( P.Dec_{sk_{P}}(e_1) ≤ P.Dec_{sk_{P}}(e_1) \)
- randomly choose \( \tilde{z} \leftarrow (0, 2^{λ+i}) \cap Z \)
- encrypt \( \tilde{z} \) by Paillier encryption to get \([\tilde{z}]\)
- output view\(_{Sim_2} = (sk_{GM}, sk_{calP}, l, [\tilde{z}],[\|\lambda\|])\)

We can get that \((sk_{GM}, sk_{P}, [\|\|],[\|\lambda\|]) = (sk_{GM}, sk_{P}, l, [\|\|],[\|\lambda\|])\). In real game, \( z = x + r \), in which \( x \) is a \( l \)-bits integer and \( r \) is a \( l + \lambda \)-bits integer, so the distribution of \( \tilde{z} \) is indistinguishable from \( z \). We can get \((sk_{P}, [\|\|]) = (sk_{P}, [\|\|])\). Moreover, since the distribution of \( z \) and \( \tilde{z} \) are independent of \( t \), so \((sk_{GM}, sk_{P}, l, [\|\|],[\|\lambda\|]) = (sk_{GM}, sk_{P}, l, [\|\|],[\|\lambda\|])\).

All in all, for all polynomial time \( A_1 \), there exists polynomial times simulator \( Sim_1 \) such that:

\[
\text{Output}_{A_1}^{\text{Real}} \approx \text{Output}_{Sim_1}^{\text{Ideal}}
\]

for all polynomial time \( A_2 \), there exists polynomial times simulator \( Sim_2 \) such that:

\[
\text{Output}_{A_2}^{\text{Real}} \approx \text{Output}_{Sim_2}^{\text{Ideal}}
\]

Therefore, we can meet the security definition defined in Definition 2, that is, the TS-RDSE scheme is \((L_1, L_2, L_3, L_4)\)-secure against adaptive chosen-keyword attacks secure in random oracle model for all polynomial time adversaries.

\section*{VII. PERFORMANCE}

\subsection*{A. THEORETICAL ANALYSIS}

In this section, we measure the performance of TS-RDSE by evaluating the complexity in terms of computation and communication overhead.

First of all, we analyze the properties of TS-RDSE and compare it with some related searchable encryption schemes that supporting search result ranking. In Table 1 we can see that, \([11]\) achieves ranking based on coordinate matching (CM). The rank method in \([14]\) is based on IPS to cluster similar files, \([22]\) designed an LSH-based scheme. However, the above rank methods only calculate the number of keywords matching in each file, which did not take into account the keyword importance. Moreover, the search results in the above schemes and in \([15], [18]\) have a certain rate of false positives, which are suitable for similarity search without providing an exact sort. TS-RDSE uses tf-idf formula to calculate the relevance weight of each keyword-file pair, and compares the encrypted weight scores based on the proven-secure
TABLE 1. Properties comparison.

| Property | Method | Security | Cryptography tool | SP | RP | Index type | Rank side |
|----------|--------|----------|-------------------|----|----|------------|-----------|
| Accuracy | [11]   | ×        | CM                | ×  | ×  | Matrix forward index | Server    |
|          | [14]   | ×        | IPS               | ×  | ×  | Inverted index      | User      |
|          | [15]   | ×        | t-idf             | ×  | ×  | Inverted index      | User      |
|          | [16]   | ×        | k-NN              | ×  | ×  | Tree forward index  | Server     |
|          | [17]   | ×        | semantic secure   | RSA/ElGamal | ×  | Forward index      | 1 Server  |
|          | [18]   | ×        | HD                | Secure | ×  | Forward index      | 2-server  |
|          | [22]   | ×        | LSH               | CKA2 | ×  | Threshold Pailler  | 2-server  |
|          | [23]   | ×        | t-idf             | CKA2 | ×  | Pailler/GM          | 2-server  |

*SP: Search Privacy; RP: Rank Privacy; CM: Coordinate Matching; IPS: Inner Product Sensitivity.

k-NN: k nearest neighbors; HD: Hamming Distance; GC: Garbled Circuits; OPE: Order-preserving Encryption.

TABLE 2. Efficiency comparison.

| Storage | Computation cost | Computation cost | Computation cost | Computation cost | Computation cost |
|---------|------------------|------------------|------------------|------------------|------------------|
| [11]    | O(NM²)           | O(qN)            | O(qd)            | O(NM²+M²)        | O(qd)            |
| [14]    | O(M²)            | O(M²)            | O(M)             | O(NM²)           | O(d)             |
| [15]    | O(NM²)           | O(qN²)           | O(qd)            | O(qgM)           | O(NM²)           |
| [16]    | O(NM²)           | O(NM²)+log(N)    | O(NM²)+log(d)    | O(NM²)+log(d)    | O(NM²)+log(d)    |
| [17]    | O(NM²)           | O(qd)            | O(qd)            | O(qd)            | O(d)             |

*Stor: Storage cost in server-side; CompGet: Computation cost for update in server-side; CompSrch: Search computation cost in server-side; CompSort: Result sorting computation cost in server-side; CompSort2: Result sorting computation cost in user-side; Comm2sort: Communication cost for user to return result.

In terms of security, [23] uses a heuristics method to hide search patterns and access patterns, so its security analysis is not within the comparison scope. [14] builds the search index based on fully homomorphic encryption (LWE-Brakerski). [13] and [14] achieve rank privacy by assuming the server just performs search and the rank is performed by user-side. The above methods reach high security but lead to high computation or communication cost. [18] and [23] also use two non-collusion two-server setting to rank the search results, however, during the interaction between servers, one of the servers can access the plaintext of each search result, which leak the search and rank privacy. TS-RDSE encrypts the weight scores based on homomorphic encryption algorithm and the servers can calculate and compare the encrypted weights without decryption, so as to sort the search results without obtaining additional information. In addition, the collaborate server S2 in TS-RDSE only plays an assistant role in the searching and sorting processes, so S2 learns nothing more than he knew already.

We also evaluate the efficiency of TS-RDSE in Table 2. Let N be file number, M be keyword number, q be the number of search keywords, qU be keyword number in an update file, and d be the file number in search results. For storage complexity, since lU only needs to store his secret key, regardless of the files number and sizes, his storage overhead is constant size. The cloud server stores the set of file ciphertexts c and the encrypted index γ, so its storage overhead of the set of file ciphertexts c is up to the size of the file sets and the encryption algorithm SK, E. For the encrypted index γ, the storage overhead is mainly determined by file number N and the keyword number M.

The token generation complexities in [11], [15] and [17] are O(qN), O(N(logN)) and O(q*N), which are all affected by the total file number and not suitable for very large dataset. In [11], the computation cost of generating a search token is O(M²). In TS-RDSE, user only needs O(q) complexity to compute the search token, which is independent with the number of files number and keywords in the cloud. Therefore the computation cost for generating search request is much lower than other schemes.

In response to the search request, the total search computation cost in [17] is O(N(logN)). And the server in [11] needs to do multiplication to compute vector products for O(qN + M²) times to perform search. In terms of TS-RDSE, two servers are required to carry out two-party protocols with the complexity of O(qd + (logd)²) for generating ranked search result with d elements, where O(qd) for searching and O((logd)²) for ranking. So the total computation complexity in TS-RDSE to search and sort is slightly higher than that of scheme [14], [17] in the case of small dataset and large search result. To get the sorted results, the schemes in [13], [14] needs user to perform rank algorithm after decrypting all the search results with complexity of O(M²) and O(qd). [15] deployed an interactive protocol between user and server to rank the result with computation cost of O(qd) and O(d).

For update, the worst update computation cost in TS-RDSE is linear with the update keyword number O(q), since update...
needs only $O(qU)$ times orthogonal vector addition and subtraction respect to the secure index, which is also not affected by the amount file number. In [17], the update computation cost for the update process is $O(M^2\log N)$, which is affected by file number and keyword number. Compared with [17] and others, TS-RDSE achieves lightweight update and can flexible extended to large dataset.

Therefore, from Table 1 and 2, we can demonstrate that TS-RDSE has both a very light user workload and a moderate server workload while being secure against honest-but curious secure model. In nowadays’s distributed cloud storage environment, how to reduce the cost of user-side storage and computation cost to ensure more flexible index update and secure server-side rank in dynamic environment is the primary concern, so TS-RDSE is more suitable for actual cloud storage deployment requirements.

B. EXPERIMENT ANALYSIS

We implement TS-RDSE in C++ under the 256 bit system security parameter. The experiments were run on linux ubuntu installed on a rotational disk. We implemented a job allocation mechanism in the server end that acts as the master server, and used threads to simulate the collaborate server that do the assistant job. As a result, our experiment environment simulates a real dual-model cloud service: to perform search, the clients only communicate with the master server, and the master server collaborate with another server to do the search job. As a comparison, we compared TS-RDSE with the scheme in [11] which also conducts multiplication of vector with matrix-based inverted index to perform keyword search. Due to the two-server ranked search is unsupported in [11], the rank method in [11] was executed by coordinate matching, so the above phase is incomparable with ours.

In the first experiment, we tested the factors that affect index storage overhead by choosing 4 different keywords dictionaries for the size of 2000/4000/6000/8000 words. The experiment results are shown in Figure 2. The abscissa is the size of the keyword dictionary, and the ordinate is the size of the storage space. It can be found that the index storage size of TS-RDSE does not change significantly with the increase of the number of keywords, because TS-RDSE uses the orthogonal vector addition method to build the index, the corresponding sub-indexes of multiple keywords are combined into one single vector.

In the second experiment, we focused on factors that affect TokenGen algorithm performance. We divided the experiment into two parts.

We first tested how different number of search keywords affected the running time of the TokenGen algorithm by choosing the keyword dictionary for the size of 2000 words and the number of search keywords as 5/10/15/20. The experiment results are shown in Figure 3. The abscissa is the number of search keywords, and the ordinate is the TokenGen time. Furthermore, we also tested how different sizes of keywords dictionary affects the TokenGen algorithm time by choosing 5 search keywords by default, and keyword dictionaries are selected as 2000/4000/6000/8000, respectively. The horizontal coordinate is the size of keyword dictionary, and the vertical coordinate is the generation time of token. It can be seen that the TokenGen running time is linear with the search keywords number, but does not change significantly as the keyword dictionary increases. This is because the length of the secret vector in the search token corresponding to the keyword is constant, it cannot significantly change the size of the keyword dictionary.

It is worth mentioning that, the algorithms can be further optimized using parallel implementation techniques which can greatly reduce the sort time at the server side, such as the multi-thread execution of sorting. However, in order to honestly reflect the original efficiency of the algorithm, we hadn’t use any optimize techniques, just simply let...
the server do all the computation in single thread each time.

**VIII. CONCLUSION AND FUTURE WORK**

In this paper, we proposed TS-RDSE—a two-server ranked dynamic searchable encryption scheme. TS-RDSE integrates homomorphic encryption cryptosystem and orthogonal vector to build a vector-level secure index, server perform simple vector multiplication between request and index to judge whether the search keywords appears. Also, the special construction of the index simultaneously supports efficient dynamic update operations like deletion and insertion of documents flexibly. In order to rank the results without decryption, we combine the widely-used tf-idf formula and partial homomorphic encryption cryptosystems to build a MPC-based secure sorting protocol, which achieves oblivious decryption, while protecting the privacy of relevance score. We give a comprehensive analysis of the correctness of TS-RDSE in the aspect of searching and sorting with formal mathematical proofs. The security analysis shows that TS-RDSE is proven $(L_1, L_2, L_3, L_4)$-secure against adaptive dynamic chosen-keyword attacks secure in random oracle model. Performance analysis and extensive experiments shows that TS-RDSE is superior to the existing approaches in terms of functionalities and suitable for the real world cloud storage environment.

As for the future work, we consider four main directions as follows:

- In TS-RDSE, adding or deleting of keywords will alter the weight scores of the entire dictionary. So our future work will be focused on designing new update methodology from the existing dictionary incurring minimum overhead on index re-construction.

- In terms of security, our vital research direction is to construct dynamic ranked SE schemes with forward and backward privacy.

- Moreover, improving rank performance in the server-side is also an important direction for future work, especially to deal with the search requests over large-scale cloud data.

**REFERENCES**

[1] Y. Zikria, S. Kim, M. Afzal, H. Wang, and M. Rehmani, “5G mobile services and scenarios: Challenges and solutions,” *Sustainability*, vol. 10, no. 10, p. 3626, 2018.

[2] C. Kelly. *Cambridge Analytica Data Millions*. Accessed: Mar. 4, 2020. [Online]. Available: https://www.cm.com/2018/04/08/politics/cambridge-analytica-data-millions/index.html

[3] T. S. Bernard, T. Hsu, N. Perlroth, and R. Lieber. *Equifax Cyberattack*. Accessed: Apr. 3, 2020. [Online]. Available: https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html

[4] MSV. Janakiram. *10 Key Takeaways From RightScale 2019*. Accessed: Apr. 3, 2020. [Online]. Available: https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html

[5] D. Xiaoding Song, D. Wagner, and A. Perrig. “Practical techniques for searches on encrypted data,” in *Proc. IEEE Symp. Secur. Privacy. (S&P)*, 2000, pp. 44–55.

[6] S. Kamara, C. Papamanthou, and T. Roeder, “Dynamic searchable symmetric encryption,” in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 965–976.

[7] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roșu, and M. Steiner, “Dynamic searchable encryption in very-large databases: Data structures and implementation,” in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.

[8] R. Bost, “Στομαχί: Forward secure searchable encryption,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1–19.

[9] Skyhigh Networks. (2016). *Skyhigh Networks: Cloud Security Software*. [Online]. Available: https://www.skyhighnetworks.com/

[10] CipherCloud. (2016). *CipherCloud: Enterprise Cloud Security*. [Online]. Available: https://www.ciphercloud.com/

[11] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, “Privacy-preserving multi-keyword ranked search over encrypted cloud data,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

[12] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, “Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing,” *IEICE Trans. Commun.*, vol. E98-B, no. 1, pp. 190–200, 2015.

[13] P. Shen, C. Chen, and X. Zhu, “Privacy-preserving relevance ranking scheme and its application in multi-keyword searchable encryption,” in *Proc. Int. Conf. Secur. Privacy Commun. Syst. Berlin, Germany: Springer*, 2014, pp. 128–146.

[14] M. Strizhov and I. Ray. “Multi-keyword similarity search over encrypted cloud data,” in *Proc. IFIP Int. Inf. Secur. Conf. Berlin, Germany: Springer*, 2014, pp. 52–65.

[15] R. Handa, C. Rama Krishna, and N. Aggarwal, “Efficient privacy-preserving scheme supporting disjunctive multi-keyword search with ranking,” *Concurrency Computat., Pract. Exper.*, vol. 32, no. 2, pp. e5450, Jan. 2020.

[16] S. Kamara and C. Papamanthou, “Parallel and dynamic searchable symmetric encryption,” in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Berlin, Germany, 2013, pp. 258–274.

[17] Z. Xia, X. Wang, X. Sun, and Q. Wang, “A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340–352, Feb. 2016.

[18] M. Li, M. Zhang, Q. Wang, S. S. M. Chow, M. Du, Y. Chen, and C. Lit, “InstantCryptoGram: Secure image retrieval service,” in *Proc. INFOCOM IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 2222–2230.

[19] S. Tahir, M. Rajarajan, and A. Sajjad, “A ranked searchable encryption scheme for encrypted data hosted on the public cloud,” in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2017, pp. 242–247.

[20] C. Wang, N. Cao, K. Ren, and W. Lou, “Enabling secure and efficient ranked keyword search over outsourced cloud data,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1467–1479, Aug. 2012.

[21] C. Wang, N. Cao, and W. Lou, “Enabling secure and efficient keyword search over encrypted cloud data,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 118–128, Jan. 2013.

[22] M. Li, T. Li, Y. Fu, X. Sun, and Q. Wang, “Comparing and improving efficiency of two-server schemes,” in *Proc. IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug. 2015, pp. 1102–1107.

[23] Z. Guan, X. Liu, L. Wu, J. Wu, R. Xu, J. Zhang, and Y. Li, “Cross-lingual multi-keyword ranked search with semantic extension over encrypted data,” *Inf. Sci.*, vol. 514, pp. 523–540, Apr. 2020.

[24] [bitglass]. (2016). *bitglass: Cloud Access Security Broker*. [Online]. Available: http://www.bitglass.com/

[25] Z. Fu, X. Wu, Q. Wang, and K. Ren, “Enabling central keyword-based semantic extension search over encrypted outsourced data,” *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 12, pp. 2986–2997, Dec. 2017.

[26] J. Wang and S. S. Chow, “Forward and backward-secure range-searchable symmetric encryption,” in *Proc. Eur. Symp. Res. Comput. Secur. Chem*, Switzerland: Springer, 2018, pp. 228–246.

[27] C. Orencik, M. Alewiwi, and E. Savas, “Secure sketch search for document ranking,” in *Proc. IEEE Trustcom/BigDataSec/ISPA*, vol. 1, Aug. 2015, pp. 1102–1107.

[28] Z. Guan, X. Liu, L. Wu, J. Wu, R. Xu, J. Zhang, and Y. Li, “Cross-lingual multi-keyword ranked search with semantic extension over encrypted data,” *Inf. Sci.*, vol. 514, pp. 523–540, Apr. 2020.

[29] CipherCloud. (2016). *CipherCloud: Enterprise Cloud Security*. [Online]. Available: http://www.bitglass.com/

[30] Z. Fu, X. Wu, Q. Wang, and K. Ren, “Enabling central keyword-based semantic extension search over encrypted outsourced data,” *IEEE Trans. Inf. Forensics Security*, vol. 17, no. 5, pp. 1591–1603, Oct. 2018.

[31] M. Etemad, A. Köpçü, C. Papamanthou, and D. Evans, “Efficient dynamic searchable encryption with forward privacy,” *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 1, pp. 5–20, Jan. 2018.

[32] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roșu, and M. Steiner, “Dynamic searchable encryption in very-large databases: Data structures and implementation,” in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.
[31] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, “Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking,” IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 11, pp. 3025–3035, Nov. 2014.

[32] Locality-Sensitive Hashing. Accessed: Apr. 5, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Locality-sensitive-hashing

[33] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, “Secure k-nearest neighbor query over encrypted data in outsourced environments,” in Proc. IEEE 30th Int. Conf. Data Eng., Piscataway, NJ, USA, Mar. 2014, pp. 664–675.

[34] B. Wang, M. Li, S. S. M. Chow, and H. Li, “A tale of two clouds: Computing on data encrypted under multiple keys,” in Proc. IEEE Conf. Commun. Netw. Secur., Oct. 2014, pp. 337–345.

[35] C. Van Ronpay, R. Molva, and M. Önen, “Fast two-server multi-user searchable encryption with strict access pattern leakage,” in Proc. Int. Conf. Inf. Commun. Secur. Cham, Switzerland: Springer, 2018.

[36] X. Meng, H. Zhu, and G. Kollios, “Top-k query processing on encrypted databases with strong security guarantees,” in Proc. IEEE 34th Int. Conf. Data Eng. (ICDE), Apr. 2018, pp. 353–364.

[37] W. Wu, U. Parampalli, J. Liu, and M. Xian, “Privacy preserving k-nearest neighbor classification over encrypted database in outsourced cloud environments,” World Wide Web, vol. 22, no. 1, pp. 101–123, Jan. 2019.

[38] K. Cheng, Y. Shen, Y. Zhang, X. Zhu, L. Wang, and H. Zhong, “Towards efficient privacy-preserving auction mechanism for two-sided cloud markets,” in Proc. IEEE Int. Conf. Commun. (ICC), May 2019, pp. 1–6.

[39] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in Proc. Int. Conf. Theory Appl. Cryptograph. Techn. Berlin, Germany: Springer, 1999, pp. 223–238.

[40] I. Damgård, M. Jurik, and J. B. Nielsen, “A generalization of Paillier’s public-key system with applications to electronic voting,” Int. J. Inf. Secur., vol. 9, no. 6, pp. 371–385, Dec. 2010.

[41] S. Goldwasser and S. Micali, “Probabilistic encryption,” J. Comput. Syst. Sci., vol. 28, no. 2, pp. 270–299, Apr. 1984.

[42] K. E. Batcher, “Sorting networks and their applications,” in Proc. Spring Joint Comput. Conf. AFIPS (Spring), vol. 2, Apr./May 1968, pp. 307–314.

[43] T. Veugen, “Improving the DGK comparison protocol,” in Proc. IEEE Int. Workshop Inf. Forensics Secur. (WIFS), Dec. 2012, pp. 49–54.

YUXI LI (Student Member, IEEE) received the M.S. degree in computer science and the Ph.D. degree in software engineering from Northeastern University, China, in 2014 and 2019, respectively, where she is currently pursuing the Ph.D. degree with the Key Laboratory of Intelligent Computing in Medical Image, Ministry of Education. Her research interests include cloud security and searchable encryption.

FUCAI ZHOU received the Ph.D. degree in computer software and theory from Northeastern University, China, in 2001. He is currently a Professor and a Ph.D. Supervisor with the Software College, Northeastern University. His research interests include cryptography, network security, trusted computing, secure cloud storage, software security, basic theory, and critical technology in electronic commerce.

ZIFENG XU received the B.S. and M.S. degrees in computer science from Bristol University, U.K., in 2011 and 2012, respectively. He is currently pursuing the Ph.D. degree with the Software College, Northeastern University, China. His research interests include verifiable computation, multi-party computation, and searchable encryption.

YUE GE received the B.S. degree in computer science from Xinyang Normal University, China, in 2017. She is currently pursuing the M.S. degree with the Software College, Northeastern University, China. She focuses on searchable encryption.

YUE GE received the B.S. degree in computer science from Xinyang Normal University, China, in 2017. She is currently pursuing the M.S. degree with the Software College, Northeastern University, China. She focuses on searchable encryption.

YUE GE received the B.S. degree in computer science from Xinyang Normal University, China, in 2017. She is currently pursuing the M.S. degree with the Software College, Northeastern University, China. She focuses on searchable encryption.

YUE GE received the B.S. degree in computer science from Xinyang Normal University, China, in 2017. She is currently pursuing the M.S. degree with the Software College, Northeastern University, China. She focuses on searchable encryption.

YUE GE received the B.S. degree in computer science from Xinyang Normal University, China, in 2017. She is currently pursuing the M.S. degree with the Software College, Northeastern University, China. She focuses on searchable encryption.

YUE GE received the B.S. degree in computer science from Xinyang Normal University, China, in 2017. She is currently pursuing the M.S. degree with the Software College, Northeastern University, China. She focuses on searchable encryption.