Scalable Fuzzy Keyword Ranked Search Over Encrypted Data on Hybrid Clouds

Hua Zhang, *Member, IEEE*, Shaohua Zhao, Ziqing Guo, Qiaoyan Wen, Wenmin Li, and Fei Gao

Abstract—Searchable encryption (SE) is a powerful technology that enables keyword-based search over encrypted data becomes possible. However, most SE schemes focus on exact keyword search which can not tolerate misspellings and typos. Existing fuzzy keyword search schemes only support fuzzy search within a limited similarity threshold d, the storage cost will grow exponentially or the precision of search results will greatly decrease as d increases. Moreover, the current fuzzy keyword ranked search schemes consider only the keyword weight, and disregard the influence of keyword morphology similarity on the ranking. In this article, we propose a scalable fuzzy keyword ranked search scheme over encrypted data under hybrid clouds architecture. We use the edit distance to measure the similarity of keywords and design an edit distance algorithm over encrypted data, in which our scheme achieves fuzzy keyword search for any similarity threshold d with a constant storage size and accurate search results. Furthermore, we design a two-factor ranking function combining keyword weight with keyword morphology similarity, which is utilized to rank the search results and enhance system usability. Extensive experiments are performed to demonstrate the trade-off of efficiency and security of the proposed scheme.

Index Terms—Searchable encryption, fuzzy keyword search, edit distance, data security, cloud computing

1 Introduction

When both enterprises and individuals outsource their data to the cloud, the data owners will encrypt their data to protect the privacy of sensitive data. It makes the effective keyword search over encrypted data become a pressing problem. Various searchable encryption (SE) schemes are proposed to resolve this problem. However, most researchers focus on exact keyword search, and the incorrect or empty search results are returned when the user inputs a misspelled query keyword. From this, fuzzy keyword search over encrypted database has been investigated and developed, which can tolerate misspells or format inconsistencies within a given similarity threshold d.

Some existing solutions have achieved accurate or efficient fuzzy keyword search over encrypted data [1], [2], [3]. However, some performances of these schemes are influenced by the similarity threshold d, such as the storage cost [1] and accuracy [2], [3]. Li *et al.* [1] adopted a wild-card approach to enumerate all similarity keywords within a predefined threshold d, and it is not scalable as the storage complexity increases exponentially with the increase of the error tolerance threshold d [2]. Subsequently, Wang *et al.* [2] and Fu *et al.* [3] enabled efficient multi-keyword fuzzy search over encrypted data with a constant size index. Their schemes both used the tool of Locality-Sensitive Hashing (LSH) for transforming the metric space on edit distance to the euclidean space, and build a pre-file index based on Bloom Filters. However, false positive and false negative exist in their schemes which influence the accuracy of the search results. All of the above schemes predefine a similarity threshold d in the setup phase, and effectual for at most two letter mistakes. For the real-world search, the similarity threshold d should be changeable according to the user's query requirement. If the length of the query keyword is shorter, the corresponding threshold d may be smaller. But, for the vocabularies used in the professional fields, keyword length is usually longer, such as medical vocabularies (pneumonoultramicroscopicsilicovolcanoconiosis, arteriosclerosis, abdominocentesis), and the gene sequences in biology. A larger threshold d is desired to tolerate more letter mistakes.

To make the user obtain the desired files quickly, it is usually necessary to rank the search results. Fu *et al.* and Ding *et al.* [3], [4] ranked the results based on keyword weight score, i.e., the value of $TF \times IDF$. This ranking method ignores the influence of keyword similarity on the ranking. The deviations between keywords and keywords will be introduced into the ranking results, and thus affects the accuracy of ranking. For example, there are three files f_1, f_2, and f_3, the keywords they contain and their corresponding weight scores are: $f_1 = \{cat, 0.5\}$, $f_2 = \{hat, 0.6\}$, $f_3 = \{bat, 0.7\}$ respectively. When the user inputs a fuzzy query request $Q = \{keyword =' cat', d = 1\}$, the cloud returns the ranked search results are $\{f_3,\ f_2,\ f_1\}$ in Fu's scheme. Apparently the expected ranked results for the user should be $\{f_1,\ f_3,\ f_2\}$. For easy of illustration, we assume here that one file only contains one keyword. In practice, in our SFRSE scheme, one file can contain many

- Hua Zhang is with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China. E-mail: zhanghua_288@bupt.edu.cn.
- Shaohua Zhao, Ziqing Guo, Qiaoyan Wen, Wenmin Li, and Fei Gao are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China. E-mail: {zhaoshaohua, guoziqing, wqy, gaof}@bupt.edu.cn, liwenmin02@outlook.com.

Manuscript received 9 Mar. 2020; revised 14 June 2021; accepted 21 June 2021. Date of publication 25 June 2021; date of current version 8 Mar. 2023. (Corresponding authors: Shaohua Zhao and Qiaoyan Wen.) Recommended for acceptance by H. Xu. Digital Object Identifier no. 10.1109/TCC.2021.3092358

keywords, the same keyword can appear in multiple files, the maximum length of the keyword is unrestricted, and the phrase can be viewed as several separate single keywords. In this paper, we introduce the keywords morphology similarity score and design a two-factor ranking function combining with keyword weight score to make the rank criterion more reasonable.

Edit distance is one of the commonly used methods to measure the keywords similarity[5]. The keywords morphology similarity score can be obtained based on the edit distance. Some existing works embedded the edit distance to Hamming distance [6] or euclidean distance [2], [3] to achieve fuzzy keyword search. However, it is proven that existing embedding approaches cannot provide sufficient distance preservation after space transformation, and will lead to false positive or false negative [7].

Edit distance calculation over encrypted data was initially implemented by secure multi-party computation and garbled circuit [5]. To decrease the computation cost, Huang *et al.* [8] optimized the garbled circuit. Cheon *et al.* [9] calculated edit distance over encrypted data with homomorphic encryption and garbled circuit, because homomorphic encryption allows for more flexible scenarios and functionality, and requires less interaction. However, Cheon's algorithm is not scalable for larger parameter due to large memory requirement. The above schemes are all based on Yao's garbled circuit. However, Yao's original construction and its variants only provide one-time security [10], whose scalability is limited.

As the main mode of cloud computing, hybrid clouds have been deployed by most cloud providers in recent years (e.g., Microsoft Azure Stack, Amazon AWS). The hybrid clouds consist of public clouds and private clouds, combining the advantage of security and controllability of the private clouds, the economy, efficiency and scalability of the public clouds [11], [12]. The public cloud can store the data owner's data and execute lots of computation with its rich storage and computing resources. The private cloud is trusted by the data owner, so that it can perform secure computing on sensitive data.

In this paper, we aim to achieve scalable fuzzy keyword ranked search over encrypted data on hybrid clouds (SFRSE). We design an edit distance algorithm over encrypted data (LDE algorithm) to verify the keywords similarity. LDE algorithm is a basic component of our SFRSE scheme, and makes our scheme implement fuzzy keyword search for any threshold d with a constant size index. The storage cost caused by a predefined threshold d can be avoided, and the limitation of threshold d can be eliminated. Thus our SFRSE scheme achieves good scalability. The clouds can compute the edit distance between two target keywords in ciphertext based on the LDE algorithm. Thus our scheme can ensure the accuracy of search results, and eliminate the false negative and false positive.

To avoid lots of unnecessary edit distance computations, we employ the idea of "filter-then-verify", and construct an index tree to improve the search efficiency. The filtering step can filter out most of the dissimilar or low similarity keywords before executing the LDE algorithm. Since we can calculate the exact edit distance and obtain keywords morphological similarity score, we combine the scores of morphological similarity with keyword weight to rank the search results. The file will be ranked ahead with a higher probability, if it contains the keywords that have higher weight score or higher morphological similarity score. The comparison among several typical fuzzy keyword SE schemes and our SFRSE scheme is shown in Table 1.

TABLE 1
Comparison of Several Typical Fuzzy Keyword SE Schemes and Our SFRSE Scheme

	[1]	[2]	[3]	SFRSE
Scalability	×	×	×	✓
False positive & false negative	×	✓	✓	×
Ranking criterion	×	KN	KW	KW & MS
Multi-keyword	×	✓	✓	×

KN, KW and MS denote keyword number, keyword weight and morphological similarity respectively.

Our contributions can be summarized as follows.

- We design a scalable fuzzy keyword ranked search scheme over encrypted data. Our scheme can support fuzzy keyword search for any similarity threshold d and long string query, and the index size is a constant that not be influenced by the increase of the threshold d.
- We realize the LDE algorithm under hybrid clouds model. With the LDE algorithm, the clouds can compute the edit distance in ciphertext and find similar keywords accurately. We construct a balanced binary tree as the index to improve search efficiency.
- We design a two-factor trade-off ranking function to rank the search results, which combines the keyword weight with morphological similarity. It makes the ranked results more in line with user expectations.

2 Preliminary

Levenshtein Edit Distance (LD) was proposed by Levenshtein in 1996 [13]. It refers to the minimum number of edit operations between two strings, which can switch one string to another. Edit distance is generally used to quantify the similarity of keywords, and is widely used in the fields of approximate string match and fuzzy keyword search. The smaller the edit distance is, the more similar the two keywords are.

The edit distance formula is presented in Equation (1), where w and q are two keywords. $ed[k][t]$ represents the edit distance of w's pre-k substring and q's pre-t substring

$$ed[k][t] = \begin{cases} t, & k = 0, t > 0 \\ k, & k > 0, t = 0 \\ min\{ed[k-1][t] + 1, ed[k][t-1] + 1, \\ ed[k-1][t-1] + f(k,t)\}, & k > 0, t > 0 \end{cases} \tag{1}$$

$$f(k,t) = \begin{cases} 0, & w[k] = q[t] \\ 1, & w[k] \neq q[t] \end{cases}.$$

Morphology Similarity Score. For any two keywords w and q, $ed(w,q)$ denotes their edit distance. Their morphology similarity score is computed by Equation (2)

$$KS_{w,q} = 1 - ed(w,q)/max\{len(w), len(q)\}. \quad (2)$$

TF-IDF rule is used to measure the relevance of a keyword in a file [14]. TF (keyword frequency) denotes the number of times a keyword appears in a file. IDF (inverse file frequency) denotes the importance of a keyword in the entire file set. There are several mathematical formulas for calculating the $TF \times IDF$ value. In this paper, we use the Equation (3) to calculate the $TF \times IDF$ value, i.e., the keyword weight score $WS_{w,f}$ of a keyword w in the file f

$$\begin{aligned} TF = N_{f,w}, \quad IDF = n/N_w \\ WS_{w,f} = (1 + lnN_{f,w}) * ln(1 + n/N_w). \end{aligned} \quad (3)$$

Here, $N_{f,w}$ denotes the keyword frequency of w_i in file f, n denotes the size of file set, N_w is the number of files containing w.

MRSE algorithm was proposed by Wang *et al.* [15], which can preserve the inner product of two vectors in ciphertext. MRSE algorithm is an enhanced algorithm of ASPE algorithm (asymmetric scalar-product-preserving encryption). The original ASPE method is vulnerable to linear analysis attack [16], and MRSE algorithm mitigates those attacks. Now, we give the main steps of MRSE scheme combining with our SFRSE scheme.

MRSE.KeyGen $(1^{\sigma}) \rightarrow K_{MRSE}$. This step is performed by the data owner. The input is a security parameter σ. The output is the key $K_{MRSE} = \{G, M_1, M_2\}$, stored by the data owner locally. The data owner randomly generates a σ-dimension binary vector $G \in \{0,1\}^{\sigma}$ and two $\sigma \times \sigma$ invertible matrices M_1 and M_2.

MRSE.EncIndex $(U_w, K_{MRSE}) \rightarrow U_w^*$. This step is performed by the data owner. The inputs are u dimension uni-gram vector U_w and the key K_{MRSE}, provided by the data owner. The output is the ciphertext U_w^*, forwarded to the public cloud. First, the data owner extends the binary vector U_w to two σ-dimension non-binary vectors $U1_w$ and $U2_w$. $U1_w = (-U_w, \; len(w), 1, \varepsilon_1, \ldots)$, $U2_w = (-U_w, 1, 1, \varepsilon_1, \ldots)$, where $\varepsilon_k, k \in [1, \; \sigma - u - 2]$ are series of random numbers. Second, the data owner extends each vector Uj_w to two random vectors Uj'_w and $Uj''_w, j = 1,2$ according to the director vector G. If $G[i] = 1$, $Uj'_w[i] = Uj''_w[i] = Uj_w[i]$. If $G[i] = 0$, $Uj'_w[i] + Uj''_w[i] = Uj_w[i], j = 1,2$. Finally, the data owner encrypts the vector U_w as $U_w^* = \{(M_1^{-1}U1'_w, M_2^{-1}U1''_w), (M_1^{-1}U2'_w, M_2^{-1}U2''_w)\}$.

MRSE.EncQuery $(U_q, K_{MRSE}) \rightarrow U_q^*$. This step is performed by the user. The inputs are the u dimension query uni-gram vector U_q and the key K_{MRSE}, provided by the user. The output is the ciphertext U_q^*, forwarded to the public cloud. First, the user extends U_q to two σ-dimension vectors $U1_q$ and $U2_q$. $U1_q = (U_q, 1, t, \theta_1, \ldots)$, $U2_q = (U_q, len(q), t, \theta_1, \ldots)$, where t is a random number and $\theta_k, k \in [1, \sigma - u - 2]$ are 0 or 1. By randomly selects V elements from $\sigma - u - 2$, the corresponding $\theta_v, v \in V$ are set to 1. Second, the user extends each vector Ui_q to two random vectors Uj'_q and $Uj''_q, j = 1,2$ according to the director vector G. If $G[i] = 1$, $Uj'_w[i] + Uj''_w[i] = Uj_w[i]$. If $G[i] = 0$, $Uj'_w[i] = Uj''_w[i] = Uj_w[i], j = 1,2$. Finally, the user encrypts the vector U_q as $U_q^* = \{(M_1U1'_q, M_2U1''_q), (M_1U2'_q, M_2U2''_q)\}$.

MRSE.Match $(U_q^*, U_w^*) \rightarrow ip(U_q^*, U_w^*)$. This step is performed by the public cloud. The inputs are two encrypted uni-gram vectors U_q^* and U_w^*. The output is the inner product $ip(U_w^*, U_q^*) = \{ip1, ip2\}$

$$\begin{aligned} ip1 = -U_w \cdot U_q + len(w_i) + t + \sum \varepsilon^v \\ ip2 = -U_w \cdot U_q + len(q) + t + \sum \varepsilon^v. \end{aligned} \quad (4)$$

Paillier Homomorphic Encryption algorithm is a kind of homomorphic encryption technique. It is secure against chosen plaintext attack (CPA) under the assumption that the decisional composite residuosity problem is hard [17].

In the Paillier cryptosystem, the public key is $PK = (n, g)$, and the private key is $SK = (\lambda, \mu)$. $n = pq$, $\lambda = lcm(p-1, q-1)$, $\mu = \lambda^{-1} \bmod n$, $g \in Z_{n^2}^*$ and $gcd((g^{\lambda} \bmod n^2 - 1)/n, n) = 1$. p and q are two independent large prime numbers chosen randomly. gcd denotes the greatest common divisor and lcm denotes the least common multiple. We use $PHE.Enc(\cdot)$ and $PHE.Dec(\cdot)$ to denote the encryption and decryption algorithm of Paillier cryptosystem respectively. For a plaintext m and its ciphertext c, the encryption and decryption process are shown as Equations (5) and (6), where r is a random number

$$PHE.Enc(m, r) = g^m \cdot r^n \bmod n^2, r \in \mathbb{R} \quad (5)$$

$$PHE.Dec(c) = \left(\frac{(c^{\lambda} \bmod n^2) - 1}{n} \right) \cdot \mu \bmod n. \quad (6)$$

Paillier cryptosystem has the properties of probability, additive and one-time multiplicative homomorphic.

For $m_1, m_2 \in Z_n, r_1, r_2, k \in Z_n^*, m_1 \neq m_2, r_1 \neq r_2$

$$\begin{aligned} PHE.Enc(m_1, r_1) \neq PHE.Enc(m_1, r_2) \\ PHE.Enc(m, r)^k = PHE.Enc(km, r) \\ PHE.Enc(m_1) \cdot PHE.Enc(m_2) = PHE.Enc(m_1 + m_2), \end{aligned}$$

where r, r_1, r_2, k are random numbers.

Pseudo-Random Permutation (PRP). $\pi : \{0,1\}^n \times \{0,1\}^s \rightarrow \{0,1\}^n$ is a PRP, if π can be computed in polynomial-time and cannot be distinguished from random function by any polynomial-time adversary [18].

3 Problem Formulation

3.1 Design Goals

To achieve scalable fuzzy keyword ranked search over encrypted data, our scheme has the following design goals.

Exactly Search Results. The proposed scheme should find out all the fuzzy keywords for a fuzzy query $Q = \{q, d\}$ exactly. The false positive and false negative are eliminated.

Similarity Threshold Scalability. In this paper, we define the "scalability" from the perspective of similarity threshold d instead of the database size. The threshold d should be scalable and should be not limited to the predefined value. The proposed scheme should achieve fuzzy keyword search for any similarity threshold d with a constant size index. The size of the index is constant regardless of the similarity threshold d, and the accuracy of search results will not be affected by the increase of d.

Two-Factor Ranking Function. To make the search results more in line with the user's expectations, the returned results should be ranked according to both keyword weight score and keyword similarity score.

Privacy-Preserving. The public cloud and the private cloud should learn no extra information except the necessary messages that appear in the entire interaction protocol. The details are described in Section 3.2.

3.2 Security Definitions

We consider two kinds of adversaries: external adversaries and internal adversaries. An external adversary can eavesdrop on information in interaction protocol, and attempt to infer privacy information as much as possible from the message he eavesdropped on. An internal adversary intends to obtain more sensitive information on the data owner's files and the user's query. In our scheme, we assume that the data owner and the user are both trusted. We assume that the public cloud and the private cloud are both "honest-but-curious". They will honestly execute the designed protocol, but they are curious about the files and query contents, and attempt to infer the privacy information by analyzing the information obtained during the protocol [2], [3], [13]. We assume that the user's input keywords are allowed to be known by the private cloud. Obviously, the internal adversary is more powerful than external adversary. In our scheme, the internal adversary may be the public cloud or the private cloud. We only discuss how to defend such an internal adversary in our scheme under the assumption that the private cloud and the public cloud are non-colluding.

SFRSE scheme should provide the following security guarantees. 1) An adversary cannot learn any extra information about the files f_i and the index keywords w_i from the encrypted file set $\mathbb{F}^*$ and the index tree $\mathcal{T}^*$. 2) The search trapdoors cannot reveal any information of the user's query keywords beyond what is implied by the search results. 3) At any time, the public cloud and the private cloud can learn only what are allowed to be leaked by the user, i.e., the search pattern and the access pattern. The search pattern and the access pattern are any information that can be derived from the search process and the access process respectively [19], and the formal definitions are as follows.

Definition 1 (Search pattern $sp(Q)$). *Given a sequence of t queries, the query history is $Query = \{Q_1, Q_2, \dots, Q_t\}$, the information that the cloud can see is $Trace = \{(T_{Q_1}, Match(T_{Q_1}, \mathcal{T}^*)), \dots, (T_{Q_t}, Match(T_{Q_t}, \mathcal{T}^*))\}$. For a query Q, the search pattern sp(Q) for Q is defined as*

$$\begin{aligned} sp(Q) = \{Q_j | \, T_{Q_j} = T_Q \text{ or } Match(T_{Q_j}, \mathcal{T}^*) \\ = Match(T_Q, \mathcal{T}^*), Q_j \in Query\}. \end{aligned}$$

T_{Q_j} and T_Q are the trapdoors of query Q_j and Q respectively. $\mathcal{T}^*$ is the encrypted index tree. $Match(T_{Q_j}, \mathcal{T}^*)$ is the match-list which contains the keywords that are match with query request Q_j. The search pattern indicates that the query Q is the same as the past query Q_j.

Definition 2 (Access pattern $ap(Q)$). *Given a fuzzy query $Q = \{q, d\}$ at time t+1, the access pattern is defined as $ap(Q) = id(Q)$. $id(Q)$ is the identifier set of files containing the keyword w_i which satisfies $ed(w_i, q) \leq d$.*

Now we use the widely-accepted simulation-based framework [19], [20], [21] to formally define the semantic security of our SFRSE scheme against adaptive attacks. The definition uses two models, real model $\mathrm{Real}_{Adv}(k)$ and ideal model $\mathrm{Ideal}_{Adv,\mathcal{S}}(k)$. In the $\mathrm{Real}_{Adv}(k)$, the challenger has the key and runs a set of prescribed algorithms to respond to the adversary's challenge. In the $\mathrm{Ideal}_{Adv,\mathcal{S}}(k)$, a simulator is defined to simulate the view of an adversary Adv during an attack. We parameterize the definition for the ideal world simulator $\mathcal{S}$ with a set of leakage functions. The leakage function captures precisely what is being leaked to the adversary during each operation [20], [21], [22]. Because our SFRSE scheme is divided into two phases: SFRSE.Setup and SFRSE.Search, we use $(\mathcal{L}^{Setup}, \mathcal{L}^{Search})$ to denote the leakage functions in Setup and Search phases respectively.

Since any adversary can only know the allowing limited leakage information but not the other information in the ideal model, we prove that the security of our SFRSE scheme by proving that the real model and the ideal model are indistinguishable for any PPT adversary. Specifically, the challenger flips a coin $coin$ at the beginning of the experiment. If $coin$=0, the challenger interacts with the adversary in the real model; otherwise interacts with the adversary in the ideal model. At the end of the experiment, the adversary outputs a bit b (0 or 1) to assert that it is in the real model or in the ideal model. Definition 3 gives the formal security definition. Equation (7) implies that the probabilities of the adversary outputs 1 are approximately equal at the two models, so the outputs of $\mathrm{Real}_{Adv}(k)$ and $\mathrm{Ideal}_{Adv,\mathcal{S}}(k)$ are indistinguishable for any adversary. $D(Real_{Adv}(k))$ and $D(Ideal_{Adv,\mathcal{S}}(k))$ denote the adversary's outputs in the real model and the ideal model, respectively. k is the security parameter, $negl(k)$ is a negligible function with k.

Definition 3 ($\mathcal{L}$-adaptively Security for SFRSE). *A SFRSE scheme is $(\mathcal{L}^{Setup}, \mathcal{L}^{Search})$-adaptively secure, if for all PPT adversaries, there exists a PPT simulator $\mathcal{S}$ such that*

$$[Pr[D(Real_{Adv}(k)) = 1]] - [Pr[D(Ideal_{Adv,\mathcal{S}}(k)) = 1]] \leq negl(k). \tag{7}$$

- *$Real_{Adv}(k)$*: Challenger runs $KeyGen(\cdot)$ to generate the key K. Adv inputs $\mathbb{F}$ and receives $(\mathbb{F}^*, \mathcal{T}^*) \leftarrow Setup.Enc(\cdot)$ from the challenger. Adv makes a polynomial number of adaptive queries $Query = \{Q_i | \, i = 1, \dots, t\}$. For each query $Q \in Query$, the challenger runs SFRSE.Search algorithm and returns the search results to Adv. Finally, Adv returns a bit b as the output of the experiment.
- *$Ideal_{Adv,\mathcal{S}}(k)$*: Adv inputs $\mathbb{F}$. Given $\mathcal{L}^{Setup}(\mathbb{F})$, $\mathcal{S}$ generates and sends $(\mathbb{F}^*, \mathcal{T}^*)$ to Adv. Adv makes a polynomial number of adaptive queries $Query = \{Q_i | \, i = 1, \dots, t\}$. For each query $Q \in Query$, $\mathcal{S}$ simulates the Search algorithm and returns the results to Adv based on $\mathcal{L}^{Search}(\mathbb{F}, Q)$. Finally, Adv returns a bit b as the output of the experiment.

4 SFRSE Scheme

Notations. Common notations and their meanings.

- $\mathbb{F}$ - The outsourced file set, $\mathbb{F} = \{f_1, \dots, f_n\}$

- $\mathbb{F}^*$ - The encryption form of $\mathbb{F}$.
- $\mathbb{W}$ - The keyword set extracted from $\mathbb{F}$, $\mathbb{W} = \{w_1, \ldots, w_m\}$
- $\mathbb{W}^*$ - The encryption form of $\mathbb{W}$.
- n - the number of files in the dataset $\mathbb{F}$.
- m - the number of keywords in the keyword set $\mathbb{W}$.
- $\mathcal{T}$ - The unencrypted index tree.
- $\mathcal{T}^*$ - The encrypted index tree.
- N - The tree node.
- $N.x$ - The element x stored in node N.
- $\mathcal{U}$ - The uni-gram element set
- Σ - The character element set.
- U_{w_i} - The 290 dimension uni-gram vector transformed from the keyword w_i.
- C_{w_i} - The L dimension character vector transformed from the keyword w_i.
- $\mathbb{U}$ - The uni-gram vector set.
- $\mathbb{C}$ - The character vector set.
- $ed(w, q)$ - The edit distance between w and q.
- $M^*(w_i, q)$ - The $L \times L$ encrypted edit distance matrix of w_i and q.
- $len(q)$ - The dimension or length of q.
- $\mathbb{VS}$ - The validation set containing the keywords which need to further verify in the verification phase.
- $\perp$ - Using in a protocol, denotes the input or output is nothing.
- PPT - Probabilistic polynomial time.
- $Z^*_{n^2}$ - Finite cyclic group of order n^2.
- gcd - The abbreviation of greatest common divisor.
- lcm - The abbreviation of lowest common multiple.
- δ_{ij} - The jth element in the vector δ_i

4.1 SFRSE Definition

As shown in Fig. 1, a SFRSE scheme consists of four entities: data owner, private cloud, public cloud and data user. The data owner is an individual or a group organization (e.g., members of a research institution), who outsource his/her encrypted dataset $\mathbb{F}^* = \{f_1^*, \ldots, f_n^*\}$ and searchable index tree $\mathcal{T}^*$ to the public cloud for data sharing. The data user is the person authorized by the data owner, who wants to access and search the dataset $\mathbb{F}^*$. The public cloud is a common commercial cloud with massive storage and computing resources, which is responsible for storing the data owner's data and processing the user's queries. The private cloud is deployed within the institution and has a partial decryption key, which is responsible for decrypting and ranking operations.

The "search control" and "access control" are performed when the data user wants to obtain authority from the data owner. The data user obtains the private key that is used to generate the query trapdoor through search control mechanisms. After executing the search protocol, the user receives the search results which are a series of ciphertext files. The data user obtains the files decryption key to decrypt and access the files through access control mechanisms. These search control and access control mechanisms can be achieved by secure channel or broadcast encryption [23], [24], which are beyond the discussion scope of this paper.

Next, we give the formal syntax of our SFRSE scheme in Definition 4. $P(x; y) \rightarrow (u; v)$ represents a protocol P that runs between two parties A and B, where x, y are the inputs, and u, v are the outputs of A and B, respectively.

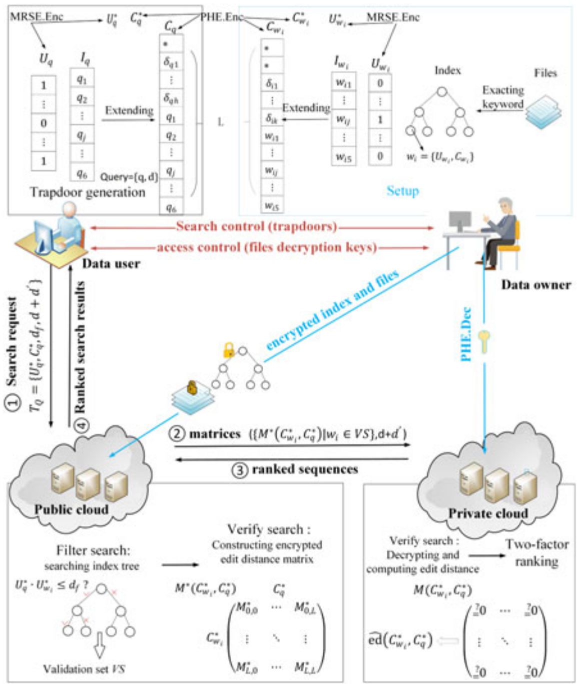

MRSE.Enc
U_q^*
C_q^*
PHE.Enc
C_q
$C_{w_i}^*$
$U_{w_i}^*$
MRSE.Enc
C_{w_i}
U_q
I_q
I_{w_i}
U_{w_i}
Extending
Extending
L
Index
Files
Exacting keyword
Query=(q, d)
Trapdoor generation
$w_i = \{U_{w_i}, C_{w_i}\}$
Setup
Search control (trapdoors)
access control (files decryption keys)
Data user
Data owner
① Search request
$T_Q = \{U_q^*, C_q^*, d_f, d + d'\}$
④ Ranked search results
encrypted index and files
PHE.Dec
② matrices $(\{M^*(C_{w_i}^*, C_q^*)|w_i \in VS\}, d+d')$
③ ranked sequences
Public cloud
Private cloud
Filter search: searching index tree
$U_q^* \cdot U_{w_i}^* \le d_f$?
Validation set VS
Verify search : Constructing encrypted edit distance matrix
$M^*(C_{w_i}^*, C_q^*)$
Verify search : Decrypting and computing edit distance
Two-factor ranking
$M(C_{w_i}^*, C_q^*)$
$\widehat{ed}(C_{w_i}^*, C_q^*)$

Fig. 1. The system model of SFRSE scheme, where red, blue and black lines represent the authority, Setup and Search phases respectively.

Definition 4 (SFRSE scheme). *A scalable fuzzy keyword ranked searchable encryption scheme consists of the following algorithms:*

SFRSE.Setup $(\mathbb{F}, 1^\mu, 1^\sigma, 1^\kappa) \rightarrow (K, \mathbb{F}^*, \mathcal{T}^*)$: *This step runs by the data owner. The inputs are a dataset $\mathbb{F}$ and three security parameters μ, σ and κ, provided by the data owner. The outputs are the key $K = (K_{Sym}, K_{MRSE}, K_{PHE} = (SK, PK))$, the encrypted dataset $\mathbb{F}^* = \{f_1^*, \ldots, f_n^*\}$ and the encrypted index tree $\mathcal{T}^*$. SK is forwarded to the private cloud. $\mathbb{F}^*$ and $\mathcal{T}^*$ are forwarded to the public cloud. The data owner publishes PK.*

SFRSE.Search $(K, Q; \mathbb{F}^*, \mathcal{T}^*; K_{PHE}) \rightarrow (\perp; \mathbb{F}_Q^*; \perp)$:*This step runs among the data user, the public cloud and the private cloud. The inputs are the key K and K_{PHE}, the fuzzy query $Q = \{q, d\}$, $\mathbb{F}^*$ and $\mathcal{T}^*$. K and Q are provided by the user. $\mathbb{F}^*$ and $\mathcal{T}^*$ are provided by the public cloud. K_{PHE} is provided by the private cloud. The output is a ranked ciphertext file set $\mathbb{F}_Q^*$, forwarded to the user.*

4.2 SFRSE Construction

SFRSE.Setup. The setup phase is performed by the data owner, and contains three steps: index generation *IndexGen*, key generation *KeyGen* and encryption *Enc*.

Step-1 Setup.IndexGen $(\mathbb{F}) \rightarrow \mathcal{T}$. The input is the dataset $\mathbb{F}$. The output is the searchable index tree $\mathcal{T}$.

Step-1.1 Extracting keywords and weight vectors $(\mathbb{F}) \rightarrow (\mathbb{W}, \mathbb{WS})$. The input is the dataset $\mathbb{F}$. The outputs are a keyword set $\mathbb{W}$ and a keyword weight vector set $\mathbb{WS} = \{WS_{w_i} | w_i \in \mathbb{W}\}$, where the dimension of WS_{w_i} is n. The data owner extracts the keywords from $\mathbb{F}$ by stemming algorithm. The ith element in weight vector WS_{w_i} is the weight score WS_{w_i, f_i} computed by Equation (3).

Step-1.2 Transforming keyword to its uni-gram vector $(w_i) \rightarrow U_{w_i}$. The input is a keyword w_i. The output is a u-dimension uni-gram vector U_{w_i}. U_{w_i} is used in filtering search step. The data owner transforms w_i to U_{w_i} based on a preset uni-gram element set $\mathcal{U}$. The set $\mathcal{U}$ is adjustable according to the specific scheme and the target dataset. In our SFRSE scheme, we choose a 290-dimension uni-gram element set $\mathcal{U} = \{a1, a2, \ldots, a10, b1, \ldots\}$ according to our experimental dataset. The set $\mathcal{U}$ contains $26 * 10$ letters, 30 numbers and commonly used symbols.

The data owner first transforms the keyword to its uni-gram set. For example, the uni-gram set of keyword $'lecture'$ is $\{l1, e1, c1, t1, u1, r1, e2\}$, where $'e2'$ indicates $'e'$ second appearance in this keyword. The data owner matches keyword to a u-dimension uni-gram vector. The element is set to 1 if the corresponding uni-gram character exists in the set $\mathcal{U}$ for the given keyword, otherwise 0. An uni-gram vector $U_{w_i} = \{0,1\}^{290}$ is generated for the keyword w_i. This transform method is proposed in reference [3].

Step-1.3 Transforming keyword to its character vector $(w_i) \rightarrow C_{w_i}$. The input is a keyword w_i. The output is a $L-$dimension character vector C_{w_i}. C_{w_i} is used in verification search step. The data owner generates the character sets Σ_1 and Σ_2 based on keyword set $\mathbb{W}$, and $\Sigma_1 \cap \Sigma_2 = \phi$. The data owner randomly chooses a dummy edit distance d' and a dummy vector δ_q, where $d' \leq min\{len(w_i)\}$. Then the data owner generates a dummy character vector set $\Delta = \{\delta_i | \delta_{ij} \in \Sigma_2, ed(\delta_i, \delta_q) = d'\}$. The data owner chooses a padding character $*$ and a fixed length L, that satisfy $L > max\{len(w_i)\} + max\{len(\delta_i)\}$ and $* \notin \Sigma_1 \cup \Sigma_2$.

The data owner first transforms keyword $w_i \in \mathbb{W}$ to a $len(w_i)$-dimension intermediate vector I_{w_i}. For example, the intermediate vector of keyword $'secure'$ is $I_{w_i} = (s, e, c, u, r, e)$. Then the data owner extends I_{w_i} to a $L-$dimension character vector C_{w_i}. The details of this operation is described in Section 5.2.1 paragraph 3-5

$$I_{w_i} = (w_{i1}, w_{i2}, \ldots, w_{ilen(w_i)}),\ w_{ij} \in \Sigma_1$$

$$C_{w_i} = (* \cdots * || \delta_i || I_{w_i}),\ \delta_i \in \Delta, w_i \in \mathbb{W}, len(C_{w_i}) = L.$$

Step-1.4 Building index $(U_{w_i}, C_{w_i}, F_{w_i}) \rightarrow \mathcal{T}$. The inputs are the tuple $(U_{w_i}, C_{w_i}, F_{w_i})$ for each $w_i \in \mathbb{W}$. The output is a balanced binary tree $\mathcal{T}$.

The elements stored in the node N of index tree $\mathcal{T}$ are

$$N = \{ID, N_l, N_r, U, C, len, WS\}.$$

- *ID* - The unique identity for each node, which is generated by a pseudorandom function denoted as *GenID()*.
- N_l - The pointer to the left child of N.
- N_r - The pointer to the right child of N.
- U - The uni-gram vector stored in N. If N is a leaf node which stores keyword w_i, $U = U_{w_i}$ is the uni-gram vector of keyword w_i. If N is an internal node, U is determined by it's left and right children

$$U[i] = OR\{N.N_l \rightarrow U[i],\ N.N_r \rightarrow U[i]\}.$$

- C - The character vector stored in N. If N is a leaf node, $C = C_{w_i}$ is the character vector of keyword w_i. If N is a internal node, C is set to $null$.
- *len* - The keyword length stored in N. If N is a leaf node which stores keyword w_i, $len = len(w_i)$ is the length of keyword w_i. If N is an internal node, len is determined by it's left and right children

$$N.len = min\{N_l.len,\ N_r.len\}.$$

- *WS* - The weight vector stored in N. If N is a leaf node which stores keyword w_i, WS is the weight vector WS_{w_i}. If N is a internal node, $WS = null$.

Step-2 Setup.KeyGen $(1^\mu, 1^\sigma, 1^\kappa) \rightarrow K$. The inputs are three secure parameters (μ, σ, κ), provided by the data owner. The output is the key $K = \{K_{Sym}, K_{MRSE}, K_{PHE}\}$. K_{Sym} is a symmetric key. $K_{MRSE} = \{S, M_1, M_2\}$ is the key of MRSE algorithm. $K_{PHE} = \{PK, SK\}$ is the key of Paillier cryptosystem. The data owner stores K locally, sends SK to the private cloud and publishes PK.

Step-3 Setup.Enc$(K, \mathbb{F}, \mathcal{T}) \rightarrow (\mathbb{F}^*, \mathcal{T}^*)$. The inputs are the dataset set $\mathbb{F}$, index tree $\mathcal{T}$ and the key K, provided by the data owner. The outputs are the encrypted dataset set $\mathbb{F}^*$ and index tree $\mathcal{T}^*$, forwarded to the public cloud. The data owner encrypts $\mathbb{F}$ to $\mathbb{F}^*$ using the key K_{Sym}, and encrypts $\mathcal{T}$ to $\mathcal{T}^*$ using the key K_{MRSE} and K_{PHE}.

When encrypting the index tree $\mathcal{T}$, the data owner encrypts each tree nodes N as N^*. The element len is embedded into the vector U^* when the data owner uses the MRSE algorithm to encrypt the vector U

$$N^* = \{ID, N_l, N_r, U^*, C^*, WS^*\},\quad C^* = PHE.Enc(C),$$
$$U^* = MRSE.EncIndex(U),\qquad WS^* = PHE.Enc(WS).$$

SFRSE.Search. The search phase is an interactive process, and runs among the user, the public cloud and the private cloud, including: trapdoor generation (*TrapGen*), filtering search (*Filter*), verification search (*Verify*) and rank (*Rank*). The inputs are the fuzzy query $Q = \{q, d\}$ and the key K, provided by the user, where q is a query keyword and d is the similarity threshold. The output is the ranked file set $\mathbb{F}^*_Q$, forwarded to the user.

Step-1 Search.TrapGen$(K, Q) \rightarrow T_Q$. This step is performed locally by the user. The inputs are the fuzzy query $Q = \{q, d\}$ and the key K, provided by the user. The output is the query trapdoor $T_Q = \{U^*_q, C^*_q, d_f, d + d'\}$, forwarded to the public cloud. d_f is the filter threshold described in detail in Section 5.1. The user transforms query keyword q to two vectors U_q and C_q, and encrypts them as U^*_q and C^*_q. The method of generating U_q is the same as the method of transforming index keyword w_i to U_{w_i}. When generating C_q, the user adds δ_q to intermediate vector I_q instead of δ_i

$$U^*_q = MRSE.EncQuery(U_q),$$
$$C^*_q = PHE.Enc(C_q) = PHE.Enc(* \cdots * || \delta_q || I_q).$$

Step-2 Search.Filter$(T_Q; \mathcal{T}^*) \rightarrow (\perp; \mathbb{VS})$. This step is performed between the user and the public cloud. The inputs are the query trapdoor T_Q and the encrypted index tree $\mathcal{T}^*$, provided by the user and the public cloud respectively. The output is a validation set $\mathbb{VS}$, forwarded to the public cloud. The public cloud searches the index tree $\mathcal{T}^*$ by the "Greedy Depth-First Search" algorithm as described in Algorithm 1, which is a recursive procedure. The specific filtering rules and search example are presented in Section 5.1.

Algorithm 1. *Search.Filter*$(T_Q; \mathcal{T}^*) \rightarrow (\perp; \mathbb{VS})$

```
Inputs: $\mathcal{T}^*$ and $T_Q$     The encrypted index tree and trapdoor,
   $T_Q = \{U_q^*, C_q^*, d_f, d + d'\}$.
Outputs: $\mathbb{VS}$     \\ The validation set
      Procedure
 1: $\mathbb{VS}$ = null      Initialize $\mathbb{VS}$ to null
 2: $root$ = root node of $\mathcal{T}^*$
 3: $t = root$  a variable node, $t = \{ID, t_l, t_r, U^*, C^*, WS^*\}$
 4: Procedure $Search(T_Q, t) \rightarrow (\perp; \mathbb{VS})$
 5: while $(t.U^*[1] \cdot U_q^*[1] \leq d_f)$ AND $(t.U^*[2] \cdot U_q^*[2] \leq d_f)$ do
 6:     if ($t_r$ = null) AND ($t_l$ = null) then
 7:         $\mathbb{VS}.append(t.C^*)$   t is a leaf node
 8:     else
 9:         $Search(T_Q, t.l) \rightarrow (\mathbb{VS})$
10:         $Search(T_Q, t.r) \rightarrow (\mathbb{VS})$
11:     end if
12: end while
13: return $\mathbb{VS}$.
```

Step-3 Search.Verify $(d + d', \mathbb{VS}; SK) \rightarrow (\perp; N_{C_q^*, d+d'})$. This step runs between the public cloud and the private cloud. The inputs are the validation set $\mathbb{VS}$, the threshold $d + d'$ and the private SK of K_{PHE}. $\mathbb{VS}$ and $d + d'$ are provided by the public cloud. SK are provided by the private cloud. The output is a set $N_{C_q^*, d+d'}$ which contains the sequence numbers of the desired fuzzy keywords, forwarded to the public cloud. In Section 5.2, we design an LDE algorithm to achieve the verification operation. In the following description, we use LDE.x to denote the x algorithm invoked from the LDE algorithm.

The public cloud constructs an $L \times L$ encrypted edit distance matrix $M^*_{C^*_{w_i}, C^*_q} = LDE.ConEDMatrix(C^*_{w_i}, C^*_q)$ for each $C^*_{w_i} \in \mathbb{VS}$, and obtains the matrix set $\mathbb{M}^*$. The public cloud generates a PRP (Pseudo-random permutation) π and sends $\pi\mathbb{M}^* = \{M^*_{C^*_{w_j}}, C^*_q \mid j = \pi i\}$ to the private cloud.

The private cloud decrypts each matrix $M^*_{C^*_{w_j}, C^*_q}$ and obtains the edit distance $ed(C^*_{w_j}, C^*_q) = LDE.ComED(M^*_{C^*_{w_j}, C^*_q})$. The private cloud stores the sequence number $j = \pi i$ to a set $N_{C_q^*, d+d'}$, which satisfies $ed(C^*_{w_j}, C^*_q) \leq d + d'$. $N_{C_q^*, d+d'} = \{j | ed\ (C^*_q, C^*_{w_j}) \leq d + d', C^*_{w_j} \in \mathbb{VS}\}$

Step-4 Search.Rank $(\mathcal{T}^*, \mathbb{F}^*; N_{C_q^*, d+d'}) \rightarrow (\mathbb{F}^*_Q; \mathbb{RS}_{C_q^*, d+d'})$. This step runs between the public cloud and the private cloud. The inputs provided by the public cloud are $\mathcal{T}^*$ and $\mathbb{F}^*$. The input provided by the private cloud is the set $N_{C_q^*, d+d'}$. The private cloud outputs the ranked file id set $\mathbb{RS}_{C_q^*, d+d'}$, and sends it to the public cloud. The public cloud outputs a ranked ciphertext file set $\mathbb{F}^*_Q$, and sends it to the user.

Step-4.1 Obtaining weight score $(\mathbb{WS}^*; N_{C_q^*, d+d'}) \rightarrow (\perp; \widehat{\mathbb{WS}}_{C_q^*, C^*_{w_j}})$. This step runs between the public cloud and the private cloud. The inputs are the encrypted weight vector set $\mathbb{WS}^*$ and the set $N_{C_q^*, d+d'}$, provided by the public cloud and the private cloud respectively. The output is the keyword weight vector set $\widehat{\mathbb{WS}}_{C_q^*, C^*_{w_j}}$, forwarded to the private cloud.

The public cloud receives set $N_{C_q^*, d+d'}$, and recovers the original sequence number $i = \pi j$ of similarity keyword w_i using the PRP π. The public cloud obtains the corresponding encrypted weight vector $WS^*_{w_j}$, further encrypts it to $\widehat{WS}^*_{w_j}$ for each $j \in N_{C_q^*, d+d'}$, and sends the set $\widehat{\mathbb{WS}}_{C_q^*, C^*_{w_j}}$ to the private cloud. This operation is to protect the weight score from the private cloud and do not impact the rank results. The private cloud decrypts each $\widehat{WS}^*_{w_j}$ as $\widehat{WS}_{w_j}$ using SK, and obtains the keyword weight vector $\widehat{WS}_{w_j}$

$$\begin{aligned}
\widehat{\mathbb{WS}}_{C_q^*, C^*_{w_j}} &= \{\widehat{WS}^*_{w_j} |\ \ j \in N_{C_q^*, d+d'}\} \\
\widehat{WS}^*_{w_j} &= (\widehat{WS}^*_{w_j, f_1}, \ldots, \widehat{WS}^*_{w_j, f_n}) \\
\widehat{WS}^*_{w_j, f_j} &= (WS^*_{w_j, f_j} \cdot PHE.Enc(r))^h \\
&= PHE.Enc((WS_{w_j, f_j} + r) \cdot h) \qquad r, h \in \mathbb{R},
\end{aligned}$$

where r and h are random numbers.

Step-4.2 Computing morphology similarity score $(N_{C_q^*, d+d'}, \mathbb{ED}) \rightarrow (\mathbb{KS}_{C_q^*, C^*_{w_j}})$. This step is performed by the private cloud. The inputs are the set $N_{C_q^*, d+d'}$ and the corresponding edit distance set $\mathbb{ED}$, provided by the private cloud. The output is the morphology similarity score set $\mathbb{KS}_{C_q^*, C^*_{w_j}}$ for each $j \in N_{C_q^*, d+d'}$

$$\begin{aligned}
\mathbb{ED} &= \{ed(C^*_q, C^*_{w_j}) \mid j \in N_{C_q^*, d+d'}\} \\
\mathbb{KS}_{C_q^*, C^*_{w_j}} &= \{KS_{C_q^*, C^*_{w_j}} | \ j \in N_{C_q^*, d+d'}\} \\
KS_{C_q^*, C^*_{w_j}} &= 1 - ed(C^*_q, C^*_{w_j}) / max\{len(C^*_q), len(C^*_{w_j})\}.
\end{aligned}$$

Step-4.3 Computing relevance score and ranking $(\widehat{\mathbb{WS}}_{C_q^*, C^*_{w_j}}, \mathbb{KS}_{C_q^*, C^*_{w_j}}; \mathbb{F}^*) \rightarrow (\mathbb{RS}_{C_q^*, d+d'}; \mathbb{F}^*_Q)$. This step runs between the private cloud and the public cloud. The inputs are set $\widehat{\mathbb{WS}}_{C_q^*, C^*_{w_j}}$, $\mathbb{KS}_{C_q^*, C^*_{w_j}}$ and $\mathbb{F}^*$, where $\widehat{\mathbb{WS}}_{C_q^*, C^*_{w_j}}$ and $\mathbb{KS}_{C_q^*, C^*_{w_j}}$ are provided by the private cloud. $\mathbb{F}^*$ is provided by the public cloud. The outputs are the ranked file id set $\mathbb{RS}_{C_q^*, d+d'}$, and the ranked file set $\mathbb{F}^*_Q$. $\mathbb{RS}_{C_q^*, d+d'}$ is forwarded to the public cloud, and $\mathbb{F}^*_Q$ is forwarded to the user.

The private cloud computes the relevance score RS_{f_i} for each $w_j, j \in N_{C_q^*, d+d'}$ as Equation (8), obtains and sends the set $\mathbb{RS}_{C_q^*, d+d'}$ to the public cloud.

$$RS_{f_i} = \sum_{i \in N_{C_q^*, d+d'}} KS_{C_q^*, C^*_{w_j}} \cdot \widehat{WS}_{w_j, f_i} \tag{8}$$

$$\mathbb{RS}_{C_q^*, d+d'} = \{id_i \mid if\ i > j, then\ RS_{f_{id_i}} > RS_{f_{id_j}}\}.$$

The public cloud finds out the corresponding encrypted file set $\mathbb{F}^*_Q$ based on $\mathbb{RS}_{C_q^*, d+d'}$, and sends it to the user

$$\mathbb{F}^*_Q = \{f_{id_i} | \ id_i \in \mathbb{RS}_{C_q^*, d+d'}\}.$$

5 The Filtering Algorithm and LDE Algorithm

5.1 Filtering Algorithm

Filtering Rules. $\mathbb{W}$ is a index keyword set and $Q = \{q, d\}$ is the user's fuzzy query. The public cloud can filter out most dissimilarity or low similarity keywords based on these values: k, d, $len(w_i)$ and $len(q)$. k is the number of matching characters between w_i and q, $k = U_{w_i} \cdot U_q, w_i \in \mathbb{W}$. $len(w_i)$ and $len(q)$ are the lengths of keywords w_i and q respectively. If $ed(w_i, q) \leq d$, the number of mismatched characters in any two keywords will not exceed d. When $len(q) - k > d$ or $len(w_i) - k > d$, it needs at least $d + 1$ edit operations to switch w_i to q. If two keywords w_i and q are similar within d, they should satisfy Equation (9)

$$len(w_i) - k \leq d \quad and \quad len(q) - k \leq d. \tag{9}$$

In our SFRSE scheme, the uni-gram vector is encrypted by the MRSE algorithm. As described in MRSE scheme of Section 2, for two uni-gram vectors U_{w_i} and U_q, their corresponding ciphertexts are $U^*_{w_i}$ and U^*_q. The inner products of the two encrypted vectors are

$$\begin{aligned} ip1 &= -U_w \cdot U_q + len(w_i) + t + \sum \varepsilon^v \\ ip2 &= -U_w \cdot U_q + len(q) + t + \sum \varepsilon^v. \end{aligned} \tag{10}$$

If the inner products of vector $U^*_{w_i}$ and U^*_q satisfy the filtering rules, i.e., the Equation (9), the corresponding inner products of the encrypted vectors should satisfy

$$ip1 \leq d + \sum \varepsilon^{(v)} + t \quad and \quad ip2 \leq d + \sum \varepsilon^{(v)} + t. \tag{11}$$

Since $\varepsilon^{(v)}$ and t are randomly chosen by the data owner and the user respectively, we set $\sum \varepsilon^{(v)} \leq 1$. To make the public cloud filter out the dissimilarity keywords, the user submits an filtering threshold $d_f = d + 1 + t$ to the public cloud.

Search Index Tree. The public cloud receives the user's query $Q = \{U^*_q, d_f\}$, and searches the index tree using the "Greedy Depth-First Search" algorithm. The public cloud calculates the inner product of $U^*_{w_i}$ and U^*_q, and returns the leaf nodes whose inner product is less than filtering threshold d_f. An example of building and searching index tree in plaintext is shown in Fig. 2.

Example: Assuming that the keyword set is $\mathbb{W} = \{w_1 :' eat', w_2 :' let',\ w_3 :' cate', w_4 :' chat', w_5 :' teach', w_6 :' tache'\}$. We generate a 6-dimension uni-gram set $\mathbb{U} = \{a, c, e, h, l, t\}$ based on the characters that appear in $\mathbb{W}$. The keywords are transformed into corresponding uni-gram vectors $U_{w_1} = (1, 0, 1, 0, 0, 1)$, $U_{w_2} = (0, 0, 1, 0, 1, 1)$, $U_{w_3} = (1, 1, 1, 0, 0, 1)$, $U_{w_4} = (1, 1, 0, 1, 0, 1)$, $U_{w_5} = (1, 1, 1, 1, 0, 1)$, $U_{w_6} = (1, 1, 1, 1, 0, 1)$. Next, we build the balanced binary index tree based on the post-order traversal method of the tree. First, we generate the leaf nodes which store the keyword uni-gram vector and keyword length. Second, we generate the parent nodes. The uni-gram vector in internal node is obtained by doing OR operation for the two uni-gram vectors stored in its child nodes, where $N.U[i] = OR\{N_l.U[i], N_r.U[i]\}$. The length stored in internal node is the minimum value of lengths stored in its two child nodes. The corresponding filtering rules in plaintext are $ip'1 = len(w_i) - k \leq d$ and $ip'2 = len(q) - k \leq d$, and the corresponding filtering threshold is $d_f = d$.

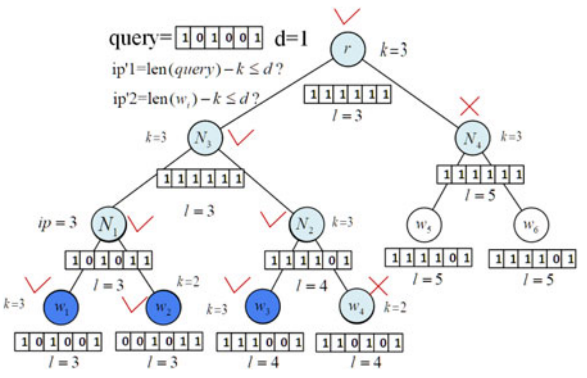


Fig. 2. An instance of building and searching index tree. k denotes the number of same characters between two vectors, l represents the keyword length, green (or gray) nodes represents the searched nodes, blue (or black) nodes represents the returned nodes, white nodes represents the unsearched nodes.

The search process is as follows. The user submits the query $Q = \{U_q = (1, 0, 1, 0, 0, 1), d_f = d = 1\}$ to the public cloud for the query keyword 'eat'. The public cloud employs the depth-first search algorithm to search the index tree. The public cloud computes the inner product $k = N.U \cdot U_q$ from the root node, and judges whether the inner product satisfies the filtering rules. If so, the public cloud continues to search its child nodes, otherwise breaks. As shown in Fig. 2, the final similarity keywords are w_1, w_2 and w_3.

5.2 LDE Algorithm

In this section, we present the design ideas and specific construction of the LDE algorithm, i.e., the verification algorithm in our SFRSE scheme.

5.2.1 Design Ideas

Transforming Keyword to its Character Vector. Edit distance algorithm is a recursive algorithm, and the ith iteration takes ith character of keywords as input to compute the substrings edit distance recursively. To make that the clouds can calculate the edit distance, we need to transform a keyword into a character vector.

Achieve LDE Algorithm Under Hybrid Clouds Architecture. As shown in Equation (1), the edit distance algorithm includes equality test and comparison test. The former is to estimate the equality of two characters ($w_i[k] \overset{?}{=} q[t]$). The latter is to find the minimum edit distance currently. In our LDE algorithm, the public cloud is responsible for constructing the encrypted edit distance matrices $M^*_{C^*_{w_i}, C^*_q}$ for each pair of target keywords $(C^*_{w_i}, C^*_q)$, $w_i \in \mathbb{VS}$. The private cloud is responsible for executing the secure computation: equality test and comparison test. These test operations will somewhat reveal the character distribution of keywords. In LDE algorithm, the public cloud sends the randomized matrix $M^*_{C^*_{w_i}, C^*_q}$ to the private cloud instead of the encrypted target keywords $C^*_{w_i}$ and C^*_q directly. This randomization operation is to prevent the private cloud from decrypting and obtaining the plaintext of query keyword and partial index keywords directly. In this way, the public cloud has the index but can not know the characters' relationship in the index keywords. The private cloud knows only the characters' equality relation of the group of similar keywords but can not know the overall character distribution information.

Adding Dummy to Protect Keyword Privacy. To avoid the keyword length and the character statistical information leaked to the clouds, we generate an intermediate vector I_{w_i} and add dummy characters to extending I_{w_i} to a L-dimension character vector C_{w_i} for each keyword w_i. On the one hand, the dimension of the intermediate vector is equal to its keyword length, and the keyword length is exposed to the two clouds. On the other hand, the private cloud executes the equality test and comparison test during search, and then it knows the co-occurrence frequency of characters and its distribution information for the keywords in the validation set. The character statistical information is disclosed to the private cloud. If the clouds have the potential adversary knowledge of the plaintexts domain (e.g., English words), they can deduce the actual plaintext keyword according to the keyword length and keyword frequency. From the view of security and privacy, such information should not be revealed to the clouds.

Due to the ordering of the edit distance algorithm, the edit distance will be affected if adding the dummy to any position of original keyword character vector randomly. The ordering refers to the element $ed[k][t]$ is determined by its previous three states $ed[k-1][t-1]$, $ed[k-1][t]$, $ed[k][t-1]$. In the LDE algorithm, we generate a dummy character vector δ_q and dummy character vector set $\Delta = \{\delta_i | \delta_{ij} \in \Sigma_2, ed(\delta_i, \delta_q) = d'\}$. We randomly choose a pair (δ_i, δ_q) and add them to the beginning of each character vector C_{w_i} and the query character vector C_q respectively. In order to ensure the accuracy of fuzzy search, the operation of adding dummy should be not change the keywords similarity, i.e., this additional operation should satisfy: if $ed(w_i, q) = d$, then $ed(\delta_i || w_i, \delta_q || q) = d + d'$.

Next, we choose a padding character $*$ and further extend the vectors $C_{\delta_i || w_i}$ to a fixed length L by adding l_p character $*$, where $l_p = L - len(\delta_i) - len(w_i)$. This step is to prevent the public cloud from learning the relative keyword lengths and deducing the exact length. Because the keyword lengths are different, the numbers of padding character to be added are different. To ensure that the addition operation does not affect the final edit distance value, we redefine the edit distance formula as Equation (12), where equation $\widehat{ed}((*\cdots* || \delta_i || w_i), (*\cdots* || \delta_q || q)) = ed(\delta_i || w_i, \delta_q || q)$ holds. The correctness analysis is described in Section 6.1

$$\widehat{ed}[k][t] = \begin{cases} 0, & k = 0, t = 0 \\ \sum_{h=0}^{t} \widehat{f}(0,h), & k = 0, t > 0 \\ \sum_{h=0}^{k} \widehat{f}(h,0), & k > 0, t = 0 \\ min\{\widehat{ed}[k-1][t] + 1, \widehat{ed}[k][t-1] \\ \quad +1, ed[k-1][t-1] + \widehat{f}(k,t)\}, & k > 0, t > 0. \end{cases} \tag{12}$$

$$\widehat{f}(k,t) = \begin{cases} 0, & w_i[k] = q[t] \\ 1, & w_i[k] \neq q[t] \end{cases}.$$

Encryption and Decryption Algorithm. To achieve character unlinkability, the encryption algorithm should be probabilistic. The edit distance algorithm involves arithmetic operations, so the LDE algorithm requires the public cloud to do some arithmetic operations in ciphertext. The encryption algorithm should be homomorphic. We choose the Paillier cryptosystem to encrypt the keyword character vectors.

5.2.2 LDE Algorithm Design

LDE.preprocess$(\mathbb{W}, \kappa) \rightarrow (\Sigma_1, \Sigma_2, d', \delta_q, \Delta, *, L, K_{PHE})$. The preprocess phase is performed by the data owner. The inputs are the keyword set $\mathbb{W}$ and security parameter κ. The output is the tuple $(\Sigma_1, \Sigma_2, d', \delta_q, \Delta, *, L, K_{PHE})$, where $K_{PHE} = \{PK, SK\}$. The data owner sends SK to the private cloud and publishes PK.

LDE.EncIndex $(w_i) \rightarrow C^*_{w_i}$. This step is performed by the data owner. The input is the keyword w_i, provided by the data owner. The output is the encrypted character vector $C^*_{w_i}$, forwarded to the public cloud.

LDE.EncQuery $(q, d) \rightarrow (C^*_q, d + d')$. This step is performed by the user. The input is a query $Q = \{q, d\}$, provided by the user. The outputs are the ciphertext C^*_q and the similarity threshold $d + d'$, forwarded to the public cloud.

The above three steps are described in Section 4.2, and we will not go into details here.

LDE.ConEDMatrix $(C^*_{w_i}, C^*_q) \rightarrow \pi M^*_{C^*_{w_i}, C^*_q}$. This step is performed by the public cloud. The inputs are two encrypted character vectors $C^*_{w_i}$ and C^*_q. The output is an encrypted edit distance matrix $\pi M^*_{C^*_{w_i}, C^*_q}$, forwarded to the private cloud, where π is a PRP.

The elements in matrix $M^*_{C^*_{w_i}, C^*_q}$ are calculated by Equation (13), where r_{kt} is a random number

$$M^*[k][t] = (C^*_{w_i}[k] / C^*_q[t])^{r_{kt}}, \quad k, t \in [0, L], \, r_{kt} \in R. \tag{13}$$

LDE.ComED $(M^*_{C^*_{w_i}, C^*_q}) \rightarrow \widehat{ed}(C^*_{w_i}, C^*_q)$. This step is performed by the private cloud. The input is the matrix $M^*_{C^*_{w_i}, C^*_q}$. The output is the edit distance. The private cloud decrypts the elements in matrix $M^*_{C^*_{w_i}, C^*_q}$ and obtains the plaintext matrix $M_{C^*_{w_i}, C^*_q}$. The private cloud calculates the edit distance $\widehat{ed}_{C^*_{w_i}, C^*_q}$ as Equation (12).

Because $M_{C^*_{w_i}, C^*_q}[k][t] = r_{kt}(w_i[k] - q[t])$ is the randomized difference between $w[k]$ and $q[t]$, we have
if $M_{C^*_{w_i}, C^*_q}[k][t] = 0$, then $w_i[k] = q[t]$,
if $M_{C^*_{w_i}, C^*_q}[k][t] \neq 0$, then $w_i[k] \neq q[t]$.

6 Correctness and Security Analysis

6.1 Correctness Analysis

Our scheme involves inner product calculation appears on *Search.Filter* step, and edit distance calculation appears on *Search.Verify* step. The correctness of the filtering algorithm relies on the correctness of the MRSE algorithm. In this section, we give the correctness analysis of our LDE algorithm.

The Correctness of Edit Distance Calculation. To prevent the private cloud from decrypting and obtaining the plain characters directly, the public cloud sends the encrypted edit distance matrix $M^*_{C^*_{w_i}, C^*_q}$ to the private cloud in *LDE.ConEncMatrix(·)*. We illustrate that this operation cannot prevent the private cloud from computing the correct edit distance in *LDE.ComED(·)*.

The element $(C^*_{w_i}[k] / C^*_q[t])^{r_{kt}}$ is the result of two corresponding position characters' arithmetic operations: two characters subtract and then multiply a random number r_{kt}. If the two characters are equal, the difference is 0, and multiply a number still is 0, so $\widehat{f}(k,t) = 0$. If the private cloud decrypts $M^*[k][t] \neq 0$, it means that the two characters are not equal, so $\widehat{f}(k,t)=1$. Thus this operation does not affect the value of $\widehat{f}(k,t)$ and the final edit distance.

The purpose of multiplying a random number is to keep the subtraction value secret for the private cloud. For example, if the private cloud knows the character $w_i[1]='a'$ beforehand, and then it obtains $M[2][2]=5$. Obviously it can further conclude $q[1]='f'$.

The Correctness of Adding Dummy. We prove that the operation of adding dummy to extend character vectors to a fixed length L will not impact the final fuzzy keyword search results. A visual example is shown in Fig. 3.

The data owner adds dummy character vector to keyword intermediate vector I_{w_i} in *LDE.EncIndex(·)*. This addition operation can be divided into two steps: 1) data owner adds $\delta_i \in \Delta$ to I_{w_i}; 2) data owner adds $L - len(C_{w_i}) - len(\delta_i)$ padding character $*$ to I_{w_i}. The operation of extending the query keyword character vector C_q in *LDE.EncQuery(·)* is similar. As shown in Figs. 3a and 3b, the way of adding dummy δ_i and δ_q satisfies

$$\begin{aligned} if: &\quad ed(I_{w_i}, I_q) = d_0, \quad ed(\delta_i, \delta_q) = d' \\ then: &\quad ed(\delta_i || I_{w_i}, \delta_q || I_q) = d_0 + d'. \end{aligned}$$

As shown in Fig. 3c, the edit distance is changed after adding $*$ to two target vectors, which is computed by the original edit distance Equation (1). To obtain the correct edit distance, we redesign the edit distance algorithm shown as Equation (12). As shown intuitively in Fig. 3d, the operation of adding padding character $*$ to two character vectors do not change the edit distance computed by Equation (12)

$$\begin{aligned} if: &\quad ed(\delta_i || I_{w_i}, \delta_q || I_q) = d_0 + d' \\ then: &\quad \widehat{ed}(C_{w_i}, C_q) = ed(* \cdots || \delta_i || I_{w_i}, * \cdots || \delta_q || I_q) \\ &\quad = d_0 + d'. \end{aligned}$$

If w_i is similar to q within threshold d, and satisfies $ed(w_i, q) \le d$, then $\widehat{ed}(C_{w_i}, C_q) \le d + d'$ must hold.

Theorem 1. *The LDE algorithm satisfies the completeness. Specially, for the user's fuzzy query request $Q = \{q, d\}$, all of the keywords w_i are returned if $ed(w_i, q) \le d$.*

Theorem 1 indicates that our SFRSE scheme can find out all desired fuzzy keywords without false positive and false negative.

The proof of *Theorem 1* is shown in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCC.2021.3092358.

6.2 Security Analysis

In this section, we give the specific contents of the leakage functions in our SFRSE scheme, and prove that our scheme is secure for any PPT adversary. The leakage function $\mathcal{L}$ is denoted as $\mathcal{L}(x_1,\ldots,x_j) = \{y_1,\ldots,y_t\}$, where $(x_1,\ldots,x_j)$ are the inputs, and $\{y_1,\ldots,y_t\}$ are the outputs.

Our SFRSE scheme is divided into two phases: SFRSE. Setup and SFRSE.Search. There exists two types of adversaries: adversarial public cloud and adversarial private cloud. We use $(\mathcal{L}_{pub}^{Setup}, \mathcal{L}_{pub}^{Search})$ and $(\mathcal{L}_{pri}^{Setup}, \mathcal{L}_{pri}^{Search})$ to denote the leakage functions in Setup and Search phases for the adversarial public cloud and the adversarial private cloud respectively.

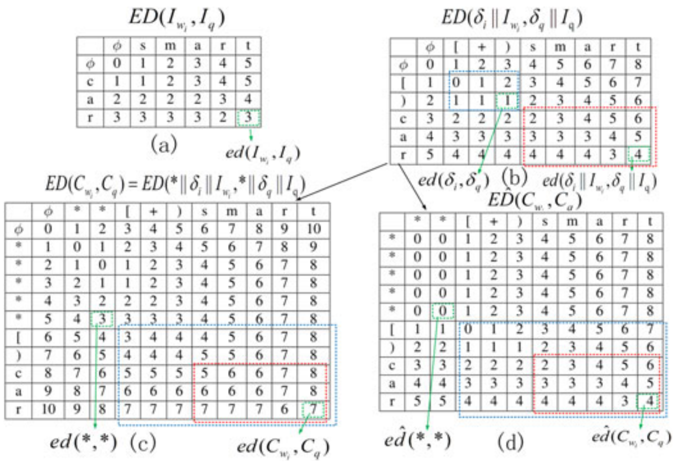

$ED(I_{w_i}, I_q)$
$ED(\delta_i || I_{w_i}, \delta_q || I_q)$
(a)
$ed(I_{w_i}, I_q)$
$ed(\delta_i, \delta_q)$
(b)
$ed(\delta_i || I_{w_i}, \delta_q || I_q)$
$ED(C_{w_i}, C_q) = ED(* || \delta_i || I_{w_i}, * || \delta_q || I_q)$
$ED(C_w, C_q)$
$ed(*,*)$
(c)
$ed(C_{w_i}, C_q)$
$\widehat{ed}(*,*)$
(d)
$\widehat{ed}(C_{w_i}, C_q)$

Fig. 3. Example of adding dummy character vectors to original keywords character vector do not change the final fuzzy keyword search results.

The Leakage Functions for the Adversarial Public Cloud.

a) Leakage function $\mathcal{L}_{pub}^{Setup}$ for setup

$$\mathcal{L}_{pub}^{Setup}(\mathbb{F}) = \{\mathbb{F}^*, \mathcal{T}^*, |\mathbb{F}|, |\mathbb{W}|, \{|f_i^*|\}_{f_i \in \mathbb{F}}, \{U_{w_i}^*, C_{w_i}^*\}_{w_i \in \mathbb{W}}\}.$$

$\mathbb{F}$ is the file set. $\mathbb{F}^*$ and $\mathcal{T}^*$ denote the encrypted file set and index tree respectively. $|\mathbb{F}|$ and $|\mathbb{W}|$ denote the number of files and keywords respectively. $|f_i^*|$ denotes the size of encrypted file f_i. $U_{w_i}^*$ and $C_{w_i}^*$ denote the encrypted unigram vector and character vector of keyword w_i.

b) Leakage function $\mathcal{L}_{pub}^{Search}$ for search

$$\mathcal{L}_{pub}^{Search}\{\mathbb{F}, Q\} = \{T_Q, sp(Q), ap(Q), \{ip1, ip2\}_{w_i \in \mathbb{W}}\}.$$

$Q = \{q, d+d'\}$ is the query. $T_Q = \{U_q^*, C_q^*, d_f, d+d'\}$ is the trapdoor. $sp(Q)$ and $ap(Q)$ are the search and access patterns of the query Q. $\{ip1, ip2\}$ are the inner products for each pair of U_q^* and $U_{w_i}^*$.

The Leakage Function for the Adversarial Private Cloud.

a) Leakage function $\mathcal{L}_{pri}^{Setup}$ for setup

$$\mathcal{L}_{pri}^{Setup}(\mu, \sigma, \kappa, \mathbb{F}) = \{K_{PHE}\}.$$

μ, σ, and κ are the security parameters. K_{PHE} is the key of paillier cryptosystem.

b) Leakage function $\mathcal{L}_{pri}^{Search}$ for search

$$\mathcal{L}_{pri}^{Search}(\mathbb{F}, Q) = \{\mathbb{M}^*, |\mathbb{M}^*|, sp(Q), \{\widehat{ed}(M_i^*), \widehat{KS}_i, WS_i\}\}.$$

$\mathbb{M}^*$ is the encrypted edit distance matrix set. $|\mathbb{M}^*|$ is the size of $\mathbb{M}^*$. $\widehat{ed}\,(M_i^*)$ is the edit distance of ith matrix in $\mathbb{M}^*$. $\widehat{KS}_i = r_i KS_i + t_i$, KS_i and WS_i are the keywords weight score and morphology similarity score respectively. r_i and t_i are random numbers.

Theorem 2. *SFRSE scheme is $\mathcal{L}_{pub}$-adaptively secure, assuming that the decisional composite residuosity problem is hard, assuming that the MRSE scheme is semantic security with indistinguishability, and assuming that the Sym is an IND-CPA secure symmetric encryption scheme. $\mathcal{L}_{pub}$ is the leakage function for the adversarial public cloud and are defined above.*

The proof is shown in Appendix B, available in the online supplemental material.

TABLE 2
Complexity Comparison of Several Typical Fuzzy Keyword Searchable Encryption Schemes With Ours

Scheme	Li *et al.* [1]		Wang *et al.* [2]		Fu *et al.* [3]		SFRSE		
Entities	User	Cloud	User	Cloud	User	Cloud	User	Public cloud	Private cloud
Storage	$O(1)$	$O(ml^d_{ave})$	$O(1)$	$O(n\tau)$	$O(1)$	$O(n\tau)$	$O(1)$	$O(m(\sigma+L))$	$O(1)$
Computation	$O(l^k_q)$	$O(ml^d_{ave}l^k_q)$	$O(\tau^2)$	$O(n\tau)$	$O(\tau^2)$	$O(n\tau)$	$O(LE+\sigma^2)$	$O(\sigma logm + \lvert\mathbb{VS}\rvert L^2E)$	$O(\lvert\mathbb{VS}\rvert L^2E)$
Communication	$O(l^k_q)$	$O(n_{q,k})$	$O(\tau)$	$O(n_{q,k})$	$O(\tau)$	$O(n_{q,k})$	$O(L+\sigma)$	$O(\lvert\mathbb{VS}\rvert L^2 + n_{q,k})$	$O(\lvert Sim_{q,k}\rvert)$

σ is the dimension of uni-gram set, U is the number of the dummy entries in MRSE algorithm, E denotes the exponential operation, L is a pre-defined length, m and n is the number of keywords and files, d is the pre-defined threshold, τ is the security parameter of EASPE. Where $\sigma = 160$, $L = 30$, $\tau = 8000$ and $d = 2$.

Theorem 3. *SFRSE shceme is $\mathcal{L}_{pri}$-adaptively secure, assuming that the PRP functions $\pi = \{\pi_i, \ldots, \pi_t\}$ is secure pseudo-random permutations. $\mathcal{L}_{pri}$ is the leakage function for the adversarial private cloud and are defined above.*

The proof is shown in Appendix C, available in the online supplemental material.

Theorem 4. *SFRSE scheme is $\mathcal{L}$-adaptive secure, if SFRSE scheme is $\mathcal{L}_{pub}$-adaptively secure for the adaptive adversarial public cloud, and $\mathcal{L}_{pri}$-adaptively secure for the adaptive adversarial private cloud.*

7 Performance Analysis

We implement our SFRSE scheme using Python 3.0 on a Windows 10 PC. The public cloud is a Tencent cloud virtual machine [25] with an AMD EPYC^{TM} 7551 CPU (2.0GHz) and 8G memory. The private cloud is a laptop with an Intel Core i7-8550 1.8GHz CPU and 8G memory. The data owner and the user both have a desktop with an Intel Core i5-3470 3.2GHz CPU and 12G memory. We use the COVID-19 Open Research Dataset [26] as our experiment dataset. We choose the key pairs of the Paillier cryptosystem is 1024-bit security, and the security parameter of MRSE algorithm is $\sigma = 450$. We choose a 290 dimension uni-gram element set $\mathcal{U} = \{a1, a2, \ldots, a10, \ldots\}$, which contains $26 * 10$ letters, 30 numbers and commonly used symbols. We set the fixed length $L = 25$ and the dummy edit distance is $d' = 1$. We estimate the overall performance of our SFRSE scheme, including the storage cost of encrypted index tree, the efficiency of index generation, the efficiency of trapdoor generation, the time of filtering search and verification search, and the rank effect of two-factor ranking. The communication latency between the public cloud and the private cloud is around 4 milliseconds on average, and we omit the latency in our performance analysis as reference [27]. The complexity comparison of several typical fuzzy keyword SE schemes with ours is shown in Table 2, including the storage, computation and communication cost in the user side and cloud side respectively.

In our SFRSE scheme, the keyword character vectors are extended to a fixed length L to protect the keywords length and characters statistical information. The security of our scheme grows with the increase of L, but the time cost also increases meanwhile. The value of L can be determined by the maximum length of the keywords in the keyword set. In our experiment dataset, the maximum keyword length is 21, so we set $L = 25$. To present the trade-off for privacy and efficiency in our SFRSE scheme, we construct a baseline scheme as the experimental comparison. The design of the baseline scheme is the same as the SFRSE scheme except that there are no dummies to be added to the keyword character vectors throughout the protocol process.

7.1 Index Generation

The process of generating an encrypted index tree $\mathcal{T}^*$ includes two stages: building an index tree $\mathcal{T}$, and encrypting the index tree. We build the index tree based on the post-order traversal method of a tree. m leaf nodes are generated based on the index keyword set $\mathbb{W}$, where m is the size of keyword set. The elements stored in each tree nodes are $\{U, C, WS\}$, whose dimension are σ, L and n, where n is the size of file set.

The process of encrypting the index tree includes two types of encryption. 1) MRSE algorithm to encrypt the uni-gram vector U. It refers to expanding vector which takes $O(\sigma)$ time, and two multiplications of two σ-dimensional vectors which takes $O(\sigma^2)$ time. 2) Paillier cryptographic algorithm to encrypt the character vector C and weight vector F. It needs an exponentiation operation E for each element in the vector. The whole time cost is $O(m(\sigma^2 + (L+n)E))$. Although it is a time-consuming process, this is a one-time operation. Once the index has been generated, the user can use it all the time until it is updated. Moreover, parallel processing technique can be used to speed up the time.

The variables that affect the time of index generation are the size of keyword set m and the size of file set n. As shown in Fig. 4, the time of index generation is linear increase with the size of keyword set m and the size of file set n. Because the time consumption of index generation is similar in the baseline scheme and the SFRSE scheme, the two time curves almost overlap. We only give the time cost of index generation for the SFRSE scheme in Fig. 4.

The space cost of storing the encrypted index tree $\mathcal{T}^*$ is $O(m(\sigma + L + n))$. The balanced binary tree $\mathcal{T}$ has space complexity $O(m)$ and each node stores two σ-dimensional vectors U^*, one L-dimensional vector C^* and one n-dimensional vector WS^*. As listed in Table 3, when the file set is fixed at n = 5000, the storage cost of $\mathcal{T}^*$ is determined by the size of keyword set m.

7.2 Trapdoor Generation

For a query request $Q = \{q, d\}$, the corresponding trapdoor is $T_Q = \{U^*_q, C^*_q, d_f, d + d'\}$. The process of trapdoor generation includes two steps. 1) Encrypting U_q with the MRSE algorithm, which takes $O(\sigma^2)$ time. 2) Encrypting C_q with the Paillier encryption algorithm, which takes $O(LE)$ time. The time cost of trapdoor generation is $O(\sigma^2 + LE)$ in the

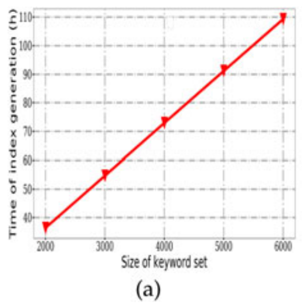


Fig. 4. Index tree generation time. a) with the size of keyword set, files number n = 5000; b) with the size of file set, keywords number m = 6000.

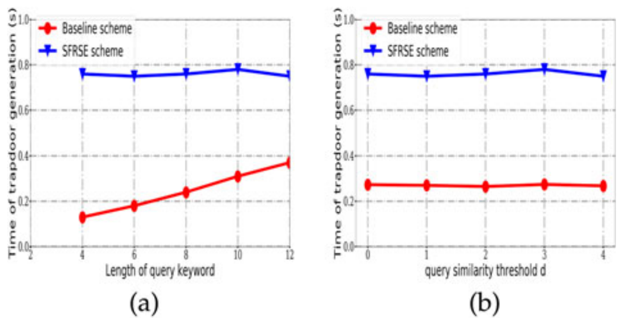


Fig. 5. Trapdoor generation time. a) with the query keyword length for a single query keyword when $d = 0$; b) with the query similarity threshold d for a specific query task “query = transfect”.

SFRSE scheme and $O(\sigma^2 + len(q)E)$ in the baseline scheme, where $len(q)$ represents the length of keyword q.

As shown in Fig. 5a, the time cost of trapdoor generation is almost constant with the increase of $len(q)$ in SFRSE scheme. The reason is that the values of σ and L are fixed in SFRSE scheme. In the baseline scheme, the length of C_q^* is $len(q)$ instead of a constant L, so the time consumption grows linearly with the increase of $len(q)$. As shown in Fig. 5b, the time of trapdoor generation is almost unaffected by the query similarity threshold d both in SFRSE scheme and baseline scheme.

7.3 Filtering Search

The public cloud receives the trapdoor, searches the encrypted index tree $\mathcal{T}^*$ and obtains the validation set $\mathbb{VS}$. This process only involves inner product computation. The time cost of computing the inner product of two σ dimension vectors U^* and U_q^* is $O(\sigma^2)$. The height of $\mathcal{T}^*$ is $log_2 m$. In the best case, the search time cost is $O(\sigma^2 log_2 m)$. In the worst case, the search time cost is $O(\sigma^2 m)$, the public cloud needs to search all leaf nodes to obtain the results.

The time cost of searching the index tree is related to the size of keyword set m and validation set $\mathbb{VS}$. On one hand, m affects the height of the index tree. On the other hand, the public cloud at least needs to search all leaf nodes in $\mathbb{VS}$ and their parent nodes. The size of $\mathbb{VS}$ depends on the specific query keyword and similarity threshold d. We estimate the time cost of searching the index tree according to a specific query task $''query = transfect''$. As shown in Fig. 6, the time of searching the index tree is almost sub-linear growth with the size of keyword set. The search time in the SFRSE scheme and the baseline scheme are approximately equal, because the dimension of U^* and U_q^* are the same in two schemes. The size of $\mathbb{VS}$ for different thresholds d and different keyword set size m are shown in Table 4.

7.4 Verification Search

The verification stage contains two steps. 1) Constructing the encrypted edit distance matrix. 2) Decrypting the matrix and calculating the edit distance. These two steps are completed by the public cloud and the private cloud separately, so we analyze their time cost respectively.

1) ConEDmatrix. The public cloud constructs the encrypted edit distance matrix $M^*_{C^*_{w_i}, C^*_q}$ for each $w_i \in \mathbb{VS}$. For each element in the matrix, the public cloud needs to perform a homomorphic addition and a multiplication operations, and we use AM to denote these operations. The dimension of matrix $M^*_{C^*_q, C^*_{w_i}}$ is $L \times L$ and $len(ave) \times len(q)$, and the time costs of constructing $|\mathbb{VS}|$ matrices are $O(L^2 AM|\mathbb{VS}|)$ and $O(len(ave)$ $len(q)AM|\mathbb{VS}|)$ in the SFRSE scheme and the baseline scheme, respectively. $len(ave)$ denotes the average keyword length in $\mathbb{VS}$, and $|\mathbb{VS}|$ denotes the size of $\mathbb{VS}$.

Since the values $|\mathbb{VS}|, len(ave)$, and $len(q)$ depend on the specific query task, we estimate the time cost according to a specific query task $''query = transfect''$. Fig. 7a shows that the time of ConEDmatrix is linear growth with the increase of the $|\mathbb{VS}|$. As shown in Fig. 7b, in the case of the query threshold $d = 0$, the time cost of ConEDmatrix is approximately the same for different sizes of keyword set. When $d = 0$, it can be viewed as an exact keyword search task, and the $|\mathbb{VS}|$ is small and almost the same for different size of keyword set, which only contains exactly query keyword and its anagram keywords (e.g., ‘atom’ and ‘moat’ are anagram keywords). Figs. 7c and 7d show that the time of ConEDmatrix increases with the increase of the threshold d.

2) ComED. The private cloud decrypts each element in the matrix $M^*_{C^*_{w_i}, C^*_q}$, and computes the edit distance. It needs L^2 and $len(ave) \times len(q)$ decryption operations for each matrix, and the time costs are $O(L^2 DE|\mathbb{VS}|)$ and $O(len(ave)len$

TABLE 3
Storage Cost of Encrypted Index Tree

size of keyword set	2000	3000	4000	5000	6000
Baseline scheme(MB)	307.79	460.71	614.25	767.81	921.41
SFRSE scheme(MB)	316.79	474.25	631.65	789.53	948.38

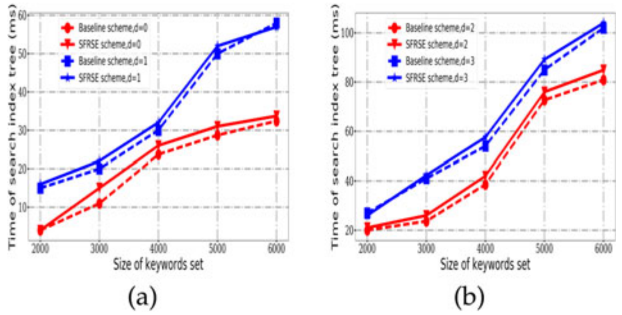


Fig. 6. Time of searching index tree with the keyword set size for different similarity threshold d, query task is “query = transfect”. a) for $d = 0, 1$; b) for $d = 2, 3$.

TABLE 4
The Size of Validation Set $\mathbb{VS}$ and Similarity Keyword Set Sim_q for Different Thresholds d and Different Keyword Set Size m, for the Query Task $''query = 'transfect''$

Size of keyword set (m)		2000	3000	4000	5000	6000
d = 0	size of $\mathbb{VS}$	1	1	1	1	1
	size of Sim_q	1	1	1	1	1
d = 1	size of $\mathbb{VS}$	1	1	3	5	8
	size of Sim_q	1	1	1	1	1
d = 2	size of $\mathbb{VS}$	6	7	7	9	12
	size of Sim_q	2	3	3	3	4
d = 3	size of $\mathbb{VS}$	48	54	59	80	101
	size of Sim_q	8	9	11	12	15

$(q)DE|\mathbb{VS}|)$ in the SFRSE scheme and the baseline scheme respectively. DE denotes the operation of homomorphic decryption.

We also estimate the total time cost according to the specific query task $''query = transfect''$. Fig. 8a shows that the time of ComED is linear growth with the increase of the $|\mathbb{VS}|$. As shown in Fig. 8b, when query threshold $d = 0$, the time consumption of ComED is approximately the same for different sizes of keyword set. The reason has been explained in the description of Fig. 7b. Figs. 8c and 8d show that the time of ComED is increasing with the increase of threshold d.

Summary. Figs. 7 and 8 both show that our SFRSE scheme consumes about six times as much time as the baseline scheme. The gap between these two schemes is larger because the fixed length L is larger than the average keyword length. This shows a trade-off in efficiency and security. The time cost in Fig. 8 is much larger than Fig. 7, because the decryption operation consumes more time than homomorphic addition and multiplication operations. When $d = 0$, the search time is in seconds, so our scheme can achieve exact keyword search efficiently. When $d = 1$, the search time is always less than 10 seconds. For the two clouds, the time consumption is larger with the increase of d, and the two clouds can use parallel processing to improve the search efficiency. For the user, our scheme can ensure that the cost of the user's computation and communication is constant for any threshold d (as shown in Fig. 5b).

7.5 Misspelled Type and Ranking Effect

Our scheme ranks the results based on keyword similarity score and keyword weight score. The experiments indicate the most relevant files will be on the top in returned results. Although the time cost of computing edit distance over encrypted data is relatively larger, it ensures that the accuracy of search results without false positives and false negatives (the proof is presented in Appendix A of Theorem 1, available in the online supplemental material). Our scheme can achieve fuzzy keyword search for any similarity threshold d, and support any misspelled types, while Fu *et al.*'s scheme cannot distinguish the anagram keywords and Wang *et al.*'s scheme cannot represent the same bi-gram [3]. The comparison of search and rank effects with several typical fuzzy keyword ranked SE schemes [2], [3] is represented in Table 5. It shows that our ranking results are more in line with the user's expectations and our scheme has good scalability.

8 Related Work

Song *et al.* [28] first proposed a practical SSE scheme. Goh *et al.* [29] proposed a formal security definition of SSE and constructed an index to improve the search efficiency. Curtmola *et al.* [19] gave an improved security definition under the adaptive adversary model. Subsequently, Cao *et al.* [15] achieved ranked multi-keyword search by the method of "coordinate matching". Xu *et al.* [30] added the keyword access frequency to rank the results and supported the index update. Sun *et al.* [47] ranked the results with the cosine similarity. Guo *et al.* [31] proposed a ranked SE scheme for multiple data owners. To enrich the search functions, Xia *et al.* [32] proposed a SSE scheme that supports dynamic update. Zhang *et al.* [33] proposed a dynamic cloud storage auditing services. Ning *et al.* [34] proposed a flexible access control scheme which could audit the correctness of outsourced decryption results. Li *et al.* [35] and Wang *et al.* [36] proposed a fine-grained SE scheme. Zhang *et al.* [37] designed a linear region search service. Recently, a series of dynamic searchable symmetric encryption schemes (DSSE) [38], [39], [40], [41] were proposed to achieve forward security against file-injection attacks [42].

Li *et al.* [1] first proposed the conception of fuzzy keyword search and employed a wildcard-based method to design their scheme, but the storage cost is growing exponentially with the increase of threshold d. Liu *et al.* [43] and Chuah *et al.* [44] improved it by limiting the fuzzy keyword's numbers or constructing a bedtree as the index, but

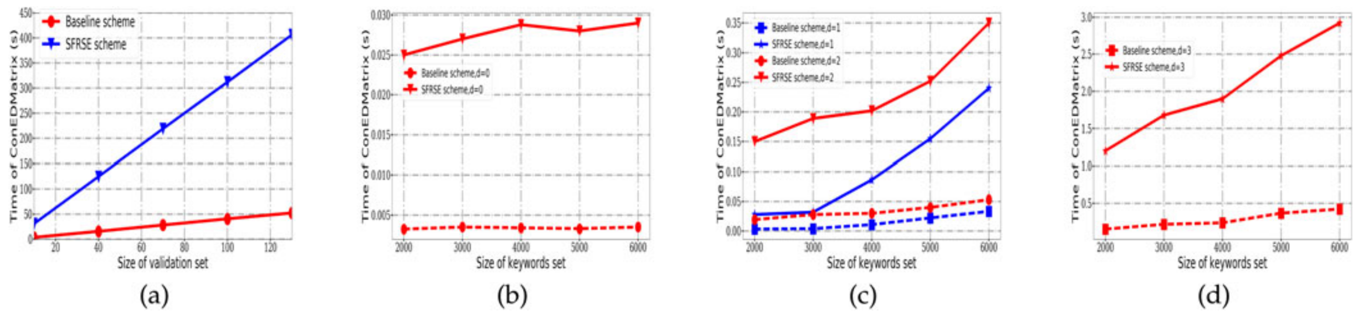


Fig. 7. Time of constructing encrypted edit distance matrix, for a specific query task "q = transfect". a) with size of validation set; b) with size of keyword set for $d = 0$; c) with size of keyword set for $d = 1$ and $d = 2$; d)with size of keyword set for $d = 3$.

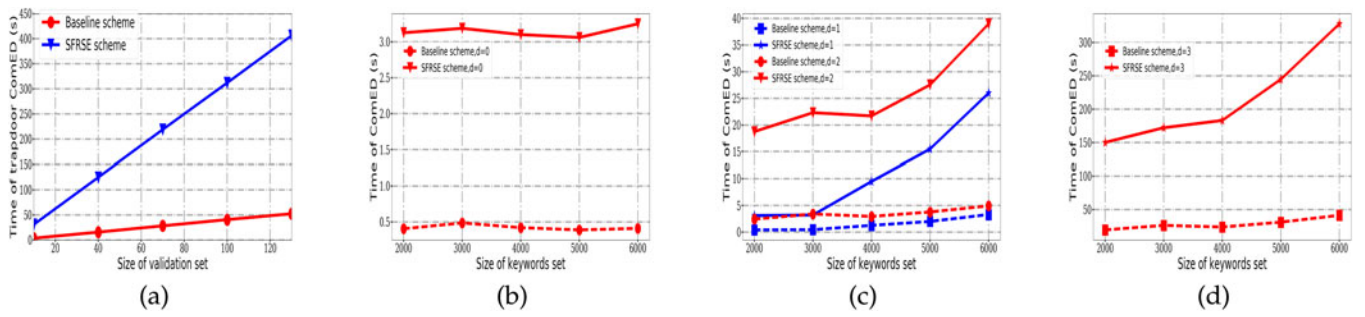


Fig. 8. Time of calculate edit distance, for a specific query task "q = transfect". a) with size of validation set; b) with size of keyword set for $d = 0$; c) with size of keyword set for $d = 1$ and $d = 2$; d)with size of keyword set for $d = 3$.

TABLE 5
Comparison of Search and Rank Effects With Several Typical Fuzzy Keyword Ranked SE Schemes

$W = \{hat, cat, atom, moat, dessert, desert, radiobroadcast, microminiaturization\}$ $f_1 = \{0.3, 0.2, 0.4, 0, 0.2, 0.6, 0, 0.8\}$ $f_2 = \{0.2, 0.2, 0, 0.6, 0.4, 0.3, 0.2, 0.3\}$ $f_3 = \{0.5, 0.1, 0.2, 0.1, 0, 0.5, 0, 0\}$ $f_4 = \{0.3, 0.7, 0.1, 0.3, 0, 0, 0.6, 0.2\}$						
	Our SFRSE scheme		Fu's scheme [3]		Wang's scheme [2]	
query(keyword,d)	similar keyword	ranking	similar keyword	ranking	similar keyword	ranking
{(atom,0)}	{atom}	f_1, f_3	{atom,moat}	f_2, f_1, f_4, f_3	{atom,moat}	disordered
{(desert,0),(atom,0)}	{desert,atom}	f_1, f_4, F_2, f_3	{dessert,desert, atom,moat}	f_2, f_1, f_3, f_4	{dessert,desert, atom,moat}	f_1, f_2, f_3, f_4
{(desert,1),(atom,1)}	{dessert,desert,atom}	f_1, f_2, f_3, f_4	{dessert,desert, atom,moat}	f_2, f_1, f_3, f_4	{dessert,desert, atom,moat}	f_1, f_2, f_3, f_4
(detsurt,2)	{dessert,desert}	f_1, f_2, f_3	{desert}	f_1, f_3, f_2	{desert}	f_1, f_2, f_3
(ratiobrodacast,3)	radiobroadcast	f_1, f_2, f_4	Null	Null	Null	Null
(micromniastsraization,4)	microminiaturization	f_4, f_2	Null	Null	Null	Null

the storage cost is still huge. Bringer *et al.* [6] achieved the fuzzy search by embedding the edit distance to Hamming distance and using the bloom filter to store index, it has a small distance distorted and false positive and false negative appears. Kuzu *et al.* [45] and Yu *et al.* [46] used a bloom filter combined with LSH to process and store all keywords extracted from files. Their schemes achieve multi-keyword search easily, but there exist false positive and false negative in their schemes. All the above schemes used the edit distance to measure the keyword similarity.

Wang *et al.* [2] and Fu *et al.* [3] used the euclidean distance as the similarity measure to achieve fuzzy multi-keyword search. Wang *et al.* transformed the keyword to biggram as the input of LSH, but is only effective for one letter mistake in a keyword. Fu *et al.* transformed keywords to uni-gram to improve the accuracy, but their scheme only support at least two letter mistakes. Fu *et al.* also enabled the ranking function by adding keyword weight to the index vector. Besides, Wang *et al.* [18] designed a similarity search scheme over feature-rich multimedia data supporting dynamic update and forward security.

9 Conclusion

In this paper, we design a scalable fuzzy keyword ranked search scheme over encrypted data on hybrid clouds. We propose a LDE algorithm which makes the clouds calculate the edit distance in ciphertext, and eliminates the limitation of query threshold d. Our SFRSE scheme ranks the search results according to the keyword weight as well as the keyword similarity score, so the ranked results are more reasonable. The performance evaluations show that our SFRSE scheme is a trade-off in efficiency and security. In the future, we will further study how to improve the search efficiency and achieve multi-keyword fuzzy search.

Acknowledgments

This work was supported in part by NSFC under Grants 62072051, 61976024, and 61972048, in part by the Fundamental Research Funds for Central Universities under Grant 2019XD-A01, and in part by the Key Project Plan of Blockchain in Ministry of Education of China under Grant 2020KJ010802.

References

[1] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. IEEE INFOCOM*, 2010, pp. 1–5.
[2] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 2112–2120.
[3] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 12, pp. 2706–2716, Dec. 2016.
[4] S. Ding, Y. Li, J. Zhang, L. Chen, Z. Wang, and Q. Xu, "An efficient and privacy-preserving ranked fuzzy keywords search over encrypted cloud data," in *Proc. Int. Conf. Behav. Econ. Socio-Cultural Comput.*, 2016, pp. 1–6.
[5] M. J. Atallah, F. Kerschbaum, and W. Du, "Secure and private sequence comparisons," in *Proc. ACM Workshop Privacy Electron. Soc.*, 2003, pp. 39–44.
[6] J. Bringer and H. Chabanne, "Embedding edit distance to allow private keyword search in cloud computing," in *Proc. FTRA Int. Conf. Secure Trust Comput., Data Manag. Appl.*, 2011, pp. 105–113.

[7] G. R. Hjaltason and H. Samet, "Properties of embedding methods for similarity searching in metric spaces," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 5, pp. 530–549, May 2003.

[8] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *Proc. USENIX Secur. Symp.*, vol. 201, no. 1, 2011, pp. 331–335.

[9] J. H. Cheon, M. Kim, and K. Lauter, "Homomorphic computation of edit distance," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2015, pp. 194–212.

[10] D. Boneh *et al.* "Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2014, pp. 533–556.

[11] B. Furht, "Cloud computing fundamentals," in *Handbook of Cloud Computing*. Boston, MA, USA: Springer, 2010.

[12] J. Li, Y. K. Li, X. Chen, P. P. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1206–1216, May 2014.

[13] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Phy. Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[14] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, 2010, pp. 253–262.

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

[16] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 733–744.

[17] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 1999, pp. 223–238.

[18] Q. Wang, M. He, M. Du, S. S. Chow, R. W. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 496–510, May/Jun. 2018.

[19] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, 2011.

[20] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2010, pp. 577–594.

[21] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 965–976.

[22] M. Du, Q. Wang, M. He, and J. Weng, "Privacy-preserving indexing and query processing for secure dynamic cloud storage," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 9, pp. 2320–2332, Sep. 2018.

[23] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[24] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 1187–1198, Apr. 2016.

[25] Tencent cloud virtual machine, cvm. Accessed: Sep. 30, 2020. [Online]. Available: https://cloud.tencent.com/product/cvm

[26] L. L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, and S. Kohlmeier, "Cord-19: The covid-19 open research dataset," 2020, *arXiv: 2004.10706*.

[27] K. Xue, S. Li, J. Hong, Y. Xue, N. Yu, and P. Hong, "Two-cloud secure database for numeric-related SQL range queries with privacy preserving," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 7, pp. 1596–1608, Jul. 2017.

[28] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, 2000, pp. 44–55.

[29] E.-J. Goh, "Secure indexes" *IACR Cryptol. ePrint Archive*, vol. 2003, p. 216, 2003.

[30] Z. Xu, W. Kang, R. Li, K. Yow, and C.-Z. Xu, "Efficient multi-keyword ranked query on encrypted data in the cloud," in *Proc. IEEE 18th Int. Conf. Parallel Distrib. Syst.*, 2012, pp. 244–251.

[31] Z. Guo, H. Zhang, C. Sun, Q. Wen, and W. Li, "Secure multi-keyword ranked search over encrypted cloud data for multiple data owners," *J. Syst. Softw.*, vol. 137, pp. 380–395, 2018.

[32] 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.

[33] L. Rao, H. Zhang, and T. Tu, "Dynamic outsourced auditing services for cloud storage based on batch-leaves-authenticated Merkle hash tree," *IEEE Trans. Serv. Comput.*, vol. 13, no. 3, pp. 451–463, May/Jun. 2020.

[34] J. Ning, Z. Cao, X. Dong, K. Liang, H. Ma, and L. Wei, "Auditable σ-time outsourced attribute-based encryption for access control in cloud computing," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 1, pp. 94–105, Jan. 2018.

[35] H. Li, Y. Yang, T. H. Luan, X. Liang, L. Zhou, and X. S. Shen, "Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 3, pp. 312–325, May/Jun. 2016.

[36] H. Wang, J. Ning, X. Huang, G. Wei, G. S. Poh, and X. Liu, "Secure fine-grained encrypted keyword search for E-Healthcare cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1307–1319, May-Jun. 2021.

[37] H. Zhang, Z. Guo, S. Zhao, and Q. Wen, "Privacy-preserving linear region search service," *IEEE Trans. Serv. Comput.*, vol. 14, no. 1, pp. 207–221, Jan./Feb. 2021.

[38] R. Bost, "οφος: Forward secure searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1143–1154.

[39] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM SIGSAC Conf. Comput Commun. Secur.*, 2017, pp. 1465–1482.

[40] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao, "Forward private searchable symmetric encryption with optimized I/O efficiency," *IEEE Trans. Dependable Comput.*, vol. 17, no. 5, pp. 912–927, Sep./Oct. 2018.

[41] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1038–1055.

[42] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. 25th Secur. Symp.*, 2016, pp. 707–720.

[43] C. Liu, L. Zhu, L. Li, and Y. Tan, "Fuzzy keyword search on encrypted cloud storage data with small index," in *Proc. IEEE Int. Conf. Cloud Comput. Intell. Syst.*, 2011, pp. 269–273.

[44] M. Chuah and W. Hu, "Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data," in *Proc. 31st Int. Conf. Distrib. Comput. Syst. Workshops*, 2011, pp. 273–281.

[45] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 1156–1167.

[46] C.-M. Yu, C.-Y. Chen, and H.-C. Chao, "Privacy-preserving multi-keyword similarity search over outsourced cloud data," *IEEE Syst. J.*, vol. 11, no. 2, pp. 385–394, Jun. 2017.

[47] W. Sun *et al.* "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proc. 8th ACM SIGSAC Symp. Inf., Comput. Commun. Secur.*, 2013, pp. 71–82.

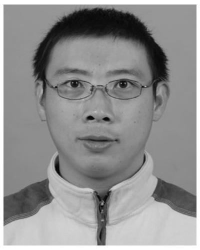
Hua Zhang (Member, IEEE) received the BS degree in telecommunications engineering and the MS degree in cryptology from Xidian University in 1998 and 2005, respectively, and the the PhD degree in cryptology from the Beijing University of Posts and Telecommunications (BUPT) in 2008. She is currently an associate professor with BUPT. Her research interests include cryptography, information security, and network security.

Shaohua Zhao received the BS degree in mathematics and applied mathematics from Henan Normal University in 2015 and the PhD degree from the Beijing University of Posts and Telecommunications. Her research interests include data security and information retrieval.

Ziqing Guo received the BS degree in mathematics and the PhD degree in computer science and technology from the Beijing University of Posts and Telecommunications in 2013 and 2019, respectively. His research interests include cloud computing security and information retrieval.

Qiaoyan Wen received the BS and MS degrees in mathematics from Shanxi Normal University, and the PhD degree in cryptography from Xidian University. She is currently the Leader of Network Security Center, Beijing University of Posts and Telecommunications. Her current research interests include cryptography, information security, and Internet security. She is a senior member of the Chinese Association for Cryptologic Research.

Wenmin Li received the BS and MS degrees in mathematics and applied mathematics from Shanxi Normal University in 2004 and 2007, respectively, and the Ph.D. degree in Cryptology from Beijing University of Posts and Telecommunications (BUPT) in 2012. She is currently an associate professor with BUPT. Her research interests include cryptography and information security.



Fei Gao received the PhD degree in cryptography from the Beijing University of Posts and Telecommunications (BUPT) in 2007. He is currently a professor with the School of network security, BUPT. His research interests include cryptography, information security, and quantum algorithm. He is a member of the Chinese Association for Cryptologic Research.