Hindawi
Security and Communication Networks
Volume 2022, Article ID 4252466, 15 pages
https://doi.org/10.1155/2022/4252466

Research Article

Verifiable Data Search with Fine-Grained Authorization in Edge Computing

Jianwei Li,[1] **Xiaoming Wang**,[1] **and Qingqing Gan**[2]

[1]*Department of Information Science and Technology, Jinan University, Guangzhou 510632, China*
[2]*Department of Cyber Security, Guangdong University of Foreign Studies, Guangzhou 510006, China*

Correspondence should be addressed to Xiaoming Wang; twxm@jnu.edu.cn

Received 6 July 2022; Revised 27 September 2022; Accepted 14 October 2022; Published 10 November 2022

Academic Editor: Jinguang Han

In the research of searchable encryption, fine-grained data authorization is a convenient way to manage the search rights for users. Recently, Liu et al. proposed a fine-grained searchable scheme with verification, which can control the search authorization and verify the results. In this paper, we first present a forgery attack against Liu et al.'s scheme and then propose a novel scheme of verifiable data search with fine-grained authorization in edge environment. Based on the key aggregate mechanism and Merkle hash tree, our proposed scheme not only achieves file-oriented search permission management but also implements the correctness and completeness verification of search results. In addition, with the assistance of edge server, resource-constrained users can easily perform the tasks of search and verification. Finally, we prove our scheme is secure based on the decision l-bilinear Diffie–Hellman exponent problem. The performance analysis and experiment results demonstrate that our proposed scheme has lower computation, communication, and storage costs contrast to the existing schemes.

1. Introduction

With the high growth of Internet technique, cloud storage and computing services have been used extensively to business and individuals [1]. While cloud services bring convenience to people, there are still many problems to be solved, such as the security and retrievability of data [2]. To satisfy these requirements, the primitive of searchable encryption (SE) [3, 4] is proposed; as a promising technology, SE allows users to search encrypted data while protecting the privacy. Traditional SE schemes are always deployed in cloud environment (such as [5–7]), which are more suitable for the users of personal computer, that the clients can deal with computationally intensive works, such as encrypting and decrypting files and configuring a large number of attributes.

In the era of mobile Internet, terminal users are developing towards diversification. In addition to computer, more and more people are using mobiles, tablets, wearables, and other devices to perceive and receive data. Constrained by limited resources, these devices cannot bear complex computations and tasks. Therefore, it is urgent to find a more friendly environment for resource-constrained terminals. Recently, edge computing has been proposed as a new paradigm [8, 9]. It configures the storage, computing, and network devices between the users and cloud, which assists users to complete tedious tasks, or sink the cloud service functions to a favorable position, providing the real-time data processing and intelligent analysis nearby. Apparently, edge computing can not only reduce the burden of terminals and decrease the service response latency but also avoid the congestion of core network. Therefore, deploying SE technology in edge environment may have wider applications.

SE technology includes symmetric encryption retrieval [3] and public key encryption retrieval [4], in which the public key encryption retrieval is mostly utilized for multiple users. In the multiuser scenario, users can share data and cooperate and communicate with each other, which is suitable for most mobile applications. For instance, in a music sharing platform, the user who publishes data is called publisher, and the user who purchases and applies data is called data user. In this system, the data users can buy the music content from publisher, and the users change dynamically, so does the purchased content. In previous studies [10–12], data authorization is user-oriented, where the users can be authorized or revoked completely. In practical applications, more fine-grained management of data authorization is required. For example, when a user purchases new content or the purchased content expires, then the publisher must distribute a new authorization or revoke the previous search rights. Therefore, file-oriented search permission management should be introduced.

In an outsourced storage environment, sometimes the user needs to deal with a malicious server. For example, the server storage data is corrupted, or the server saves computing resources during the peak time. In these situations, the server may not want to process the whole database when responding to queries. From the perspective of users, they pay for the data and then always expect to receive guaranteed services. Therefore, the function of data verification for user is required when necessary. Recently, some schemes (e.g., [13, 14]) have been proposed to verify the search results. However, these schemes can only verify the correctness of return data but not the completeness. If the cloud returns an insufficient number of files, the user cannot discover it. Later, Liu et al. [15] proposed a searchable scheme to verify the completeness of search results. Unfortunately, this scheme involves forgery attack so that the users cannot correctly verify the completeness of search results.

In this paper, we propose an efficient verifiable data search with file-oriented authorization scheme in edge computing, in which the publishers can distribute fine-grained search permissions, and the data users can search the favorite data and verify the correctness and completeness of results with the assistance of edge server. Our contributions are as follows:

(i) We analyze and show an attack against Liu et al.'s scheme [15].

(ii) We propose a novel privacy-preserving file-oriented search scheme in edge computing. Based on the key aggregate mechanism, our proposed scheme implements fine-grained authorization and facilitates the distribution of massive data search rights. Through the design of Merkle hash tree, it achieves the correctness and completeness verification of search results. In addition, with the assistance of edge server, the resource-constrained users can easily query the data and verify the search results.

(iii) We optimize the costs of the proposed scheme. In the keyword ciphertext process, it uploads one Bloom filter value instead of all the encrypted keywords, and then the communication and storage costs are related to the number of files, instead of the keyword number.

(iv) Based on the decision l-BDHE problem, we prove our scheme can meet the secure features. Performance evaluation and experiments demonstrate that our scheme is more practical and efficient than the available schemes.

The rest paper is organized into seven sections. The related studies and reviews are described in Section 2. The relevant knowledge is introduced in Section 3. Section 4 discusses the attack of Liu et al.'s scheme. Section 5 states the details of our proposed scheme. Then we show the requirements analysis in Section 6 and the performance evaluation in Section 7. The last section is a summary.

2. Related Work

To address the issues of data searchability and privacy preserving, Song et al. [3] proposed the primitive of searchable symmetric encryption, which can search the data with encryption form. Later, Boneh et al. [4] proposed the Public Key Searchable Encryption (PKSE) and applied it to mail system. Since then, PKSE has become a research hotspot, and many solutions have been proposed such as the proxy reencryption PKSE [16], attribute-based PKSE [17], certificateless PKSE [18], and PKSE based on primes [19]. However, these schemes are suitable for the personal computer clients, which are computation-intensive and unfriendly to resource-constrained terminals.

To reduce the computing and storage overhead in the client side, Guo et al. [20] proposed a keyword search encryption framework under the edge environment, which offloads the computation-intensive tasks of the sensor to the edge server. However, they only propose a framework without a specific implementation. Chen et al. [21] presented a privacy-preserving searchable encryption scheme in the edge computing, which designs an S-HashMap index structure and supports the fuzzy search of multikeywords. Yet in their scheme, the user needs to calculate all keyword indexes and generate index access tree, which requires a large amount of computation. Scheme [22] puts forward a PKSE solution based on the witness system in cloud-edge computing to resist the keyword guessing attack, while the scheme can only resist the external attackers but not the internal attackers like the curious cloud server. Wang et al. [23] proposed an image retrieval scheme with mobile edge computing, which introduced a cloud-guided image feature extraction method, thus reducing network traffic and improving retrieval accuracy, but this solution cannot be applied to keyword search scenario. Scheme [24] proposed a user-centered data search framework, which uses the edge computing to make intelligent prediction of the user's search pattern, tailoring the search space so as to reduce the processing time. However, the framework is user-centered and is not suitable for the numerous files sharing scenario.

In the scenes of numerous searchable files sharing (such as [25–27]), the data owner can distribute the file-oriented retrieval rights to other users. In these schemes, the data owner uses different keys to encrypt different documents and shares the corresponding keys to the legitimate user. Obviously, such a key management mechanism has high cost in aspects of communication and storage, and the size of search token increases with the number of share files. Scheme [28] introduced a proxy server to transform the user's search token into a instance, so as to reduce the key cost of user side. However, this method does not fundamentally solve the problem of inefficient key management. To solve the key management overhead problem, Cui et al. [29] presented a key aggregate searchable encryption in the cloud storage environment. In this scheme, the client can search for multiple files with one single key shared by the data owner. However, Zhou et al. [30] referred that the scheme [29] suffered from keyword guessing attack. Later, Li et al. [31] proposed a multiowner key aggregate searchable encryption scheme that the user can submit one single trapdoor to search across multiple data owners' file records. Yet the scheme does not consider the dynamic search right management.

For the verification mechanism, Zhang et al. [32] proposed that a scheme can verify the single keyword search ranking results of the data uploaded by multiple users. However, all the users need to share the sorting data interactively to obtain the final ranking order, which reduces the practicability. Jianfeng Wang et al. [33] proposed a method to verify the query results of multiple keywords, which is suitable for the large scalable database, but this scheme uses an accumulator to implement verification and makes it difficult for mobile clients to bear the computing overhead. Recently, schemes [13, 14] have been proposed to verify the aggregate search results, but these schemes can only verify the correctness but not the completeness. A scheme [15] is proposed which can verify the completeness of results; however, it suffers from forgery attack so that the users cannot correctly verify the completeness of search results.

3. Preliminaries

3.1. Bilinear Pairing. Let two multiplicative cyclic groups $\mathbb{G}$ and $\mathbb{G}_1$ have the identical order p, bilinear pairing e is a map: $\mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_1$ which has the attributes as follows:

(1) Bilinearity: For $a, b \in \mathbb{G}$ and $m, n \in \mathbb{Z}_P$, we have $e(a^m, b^n) = e(a, b)^{mn}$

(2) Computability: There is an efficiency algorithm to calculate $e(a, b)$ for any $a, b \in \mathbb{G}$

(3) Nondegeneracy: Let ϑ be a generator of $\mathbb{G}$, then $e(\vartheta, \vartheta) \neq 1$

3.2. Merkle Hash Tree. The Merkle hash tree (MHT) is a data structure based on the hash function which is a hash binary tree. In MHT, each leaf node stores the hash value of the data block, and each nonleaf node's value is calculated by hashing its children.

In MHT construction, we first calculate the hash value of each data, and then fill them as the lowest leaf nodes. Next, we establish the upper layer, and the value of each node in this layer is calculated by hashing its left and right children. Then we continuously establish the upper layer until the root.

MHT is recursively calculated layer by layer through hash computation, while hash is a one-way function; that is, the value of the parent node can only be calculated by its children, and the value of the child node cannot be deduced from the parent node. Therefore, when the value of the root node is determined, the correctness of all other nodes' value can be guaranteed, namely the change of any node's value will cause the value of root node to be different.

For example, a data set $D = \{d_1, d_2, d_3, d_4\}$ is input, then the output of MHT (D) is the root node's value L_{root}, then we show the construction in Figure 1. First, we compute the hash value $L_i = H(d_i)_{i=\{1,2,3,4\}}$ for each leaf node, then calculate the nonleaf nodes' values $L_5 = H(L_1 L_2)$ and $L_6 = H(L_3 L_4)$, and at last, we output the root node' value $L_{\text{root}} = H(L_5 L_6)$.

3.3. Bloom Filter. Bloom filter is a data structure used to represent collections, and it has three operations as follows:

(1) BFInit: This operation creates an empty Bloom filter, which is a l-bit array with value 0

(2) BFAdd$(\{H_1, \ldots, H_k\}, bf, s)$: This operation adds an element s to the Bloom filter bf. It first hashes the element with k functions as $H_i(s) = \varphi_i \in \{0, 1, \ldots, l-1\}$, and then sets the φ_i-th bit value to 1

(3) BFQuery$(\{H_1, \ldots, H_k\}, bf, s)$: This operation queries whether the element s is a membership data. It first computes the k hash functions to s, and then checks whether all the corresponding bits are equal to 1. If it holds true, s is a membership data; otherwise, s is not in the collection.

3.4. Keyed Hash Function. The keyed hash function is a verification and encryption mechanism used by communication entities, and it ensures the integrity and confidentiality of message data, and its security depends on the hash function. The keyed hash function inputs a message and a key and then outputs a hash value used for data authentication and integrity verification.

In this paper, we use the keyed hash function HMAC [34] to process keywords, and the calculation method is as follows:

$$H_E(m) = H(E \oplus \text{opad} \| H(E \oplus i\text{padm}) \|), \tag{1}$$

where m is a message, E is the key which is a 64 bits string, H is a hash function, opad and ipad are strings composed of several "$0x5c$" and "$0x36$," respectively, $\oplus$ represents XOR operation, and $\|$ represents join operation.

3.5. Complexity Assumption. The complexity assumption is defined as follows:

(1) Decision l-bilinear Diffie–Hellman exponent (Decision l-BDHE) problem: The problem [35] in group $\mathbb{G}$ is worked as below. Given a vector of $2l + 2$ elements $(h, \vartheta, \vartheta^a, \vartheta^{a^2}, \ldots, \vartheta^{a^l}, \vartheta^{a^{l+2}}, \ldots, \vartheta^{a^{2l}}, Z)$ as input, where $h, \vartheta \in \mathbb{G}$, $Z \in \mathbb{G}_1$, and ϑ^{a^i} for $i = \{1, 2, \ldots, l, l+2, \ldots 2l\}$, note that $\vartheta^{a^{l+1}}$ is missing, and a had not given. Then decide if $Z = e(\vartheta, h)^{a^{l+1}}$ or Z is a random value in $\mathbb{G}_1$. For a polynomial time

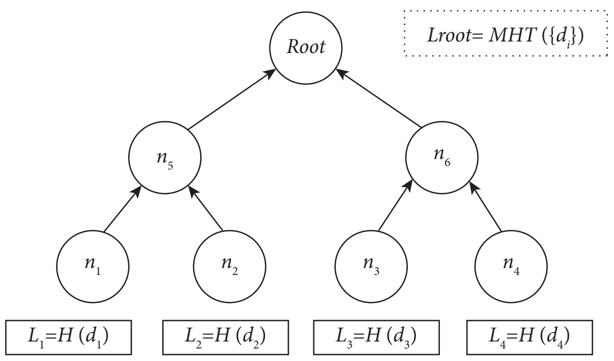


Figure 1: Merkle hash tree.

adversary Ad, the advantages obtained from the problem are defined as

$$ADV_{Ad}^{\text{Decisionl-BDHE}} = \left| \begin{array}{l} \Pr\left[A\left(h, \vartheta, \vartheta^{a}, \vartheta^{a^2}, \ldots, \vartheta^{a^l}, \vartheta^{a^{l+2}}, \ldots, \vartheta^{a^{2l}}, Z = e(\vartheta, h)^{a^{l+1}}\right) = 0\right] \\ -\Pr\left[A\left(h, \vartheta, \vartheta^{a}, \vartheta^{a^2}, \ldots, \vartheta^{a^l}, \vartheta^{a^{l+2}}, \ldots, \vartheta^{a^{2l}}, Z \neq e(\vartheta, h)^{a^{l+1}}\right) = 0\right] \end{array} \right|. \quad (2)$$

(2) Decision l-BDHE assumption: For the adversary Ad with any polynomial time t, the advantage to Decision l-BDHE problem is defined as $ADV_{Ad}^{\text{Decision}l\text{-}BDHE} \leq \varepsilon$, then we say that the (t, ε, l)-BDHE problem is difficult to be solved.

4. Discuss of Liu et al.'s Scheme

4.1. Liu et al.'s Scheme. We simply describe Liu et al.'s scheme [15] as follows:

(1) Init $(1^{\lambda}, n)$: The system initializes parameters described below.

(a) Generates a bilinear map group system $\mathbb{B} = (p, \mathbb{G}, \mathbb{G}_1, e(\cdot,\cdot))$ and sets n as the maximum number of documents

(b) Randomly picks $g \in \mathbb{G}$ and $a \in \mathbb{Z}_p$ and computes $g_i = g^{a^i}$, for $i = 1, 2, \ldots, n, n+2, \ldots 2n$

(c) Selects one-way hash functions H_0: $\{0,1\}^* \longrightarrow \mathbb{G}$, H_1: $\mathbb{G}_1 \longrightarrow \{0,1\}^m$ and $H_1', \ldots, H_k'$: $\{0,1\}^* \longrightarrow \{0, \ldots, m-1\}$

(2) KeyGen: The data owner chooses a random $\gamma \in \mathbb{Z}_p$ and sets the private key $sk = \gamma$ and public key $v = g^{\gamma}$

(3) Encrypt (i, W_i): For the i-th document $(i \in = \{1, \ldots, n\})$, the data owner does those as given below.

(a) Generates a Bloom filter for this document's keyword set by computing $BF_i = \text{BFGen}$ $(\{H_1', \ldots, H_k'\}, W_i)$

(b) Picks a random $t \in \mathbb{Z}_p$, $M \in \mathbb{G}_1$ and then computes the ciphertexts and sends to cloud: $(c_1, c_2, c_3, c_4) = (g^t, (v \cdot g_i)^t, H_1(M) \oplus BF_i, M \cdot e(g_1, g_n)^t)$ and $c_w = e(g, H_0(w))^t / e(g_1, g_n)^t$

(4) Share (S): For subset $S \subseteq \{1, \ldots, n\}$, the data owner computes $K_a = \prod_{j \in S} g_{n+1-j}^{\gamma}$ and sends to user

(5) Trapdoor (w): The user generates the trapdoor $\text{Tr} = K_a \cdot H_0(w)$ and sends to the cloud

(6) Retrieve (Tr, S):

(a) The cloud tests the trapdoor for all files in S, and Test: $cw = e(\text{Tr} \cdot \prod_{j \in S, j \neq i} g_{n+1-j+i}, \ c_1) / e \ \ (\prod_{j \in S} g_{n+1-j}, c_2)$

(b) The cloud generates the verification proofs as

$$\text{proof}_i = (p_1, p_2, p_3) = \frac{\left(c_4 \cdot \prod_{j \in S, j \neq i} g_{n+1-j+i}, c_1\right)}{e\left(\prod_{j \in S} g_{n+1-j}, c_2\right)}, c_1, c_3. \quad (3)$$

(7) Verify (w, proof): The user verifies the results as follows:

$$M' = p_1 \cdot e(K_a, p_2), BF_i' = H_1(M') \oplus p_3, BF\text{verify} \cdot (\{H_1', \ldots, H_k'\}, BF_i', w) = 1. \quad (4)$$

4.2. Analysis. Liu et al.'s scheme realizes that an authorized user can retrieve multiple encrypted files by one shared key and verifies the completeness of search results, but we find that the user cannot correctly verify the completeness of search results. In their scheme, the verification proof is calculated and transmitted by the cloud, and then it has the opportunity to forge the proof, which the users could not detect.

Let us look at a specific case. Suppose authorized subset $S = \{f_1, f_2, \ldots, f_s\}$, with the user queries keyword w, then the honest cloud will return the complete search results $I = (\{f_1, f_2, \ldots, f_t\})$, $(t \le s)$, and the verification proofs are

$$\text{proof}_i = (p_1, p_2, p_3) = \frac{\left(c_4 \cdot \prod_{j\in S, j\neq i} g_{n+1-j+i}, c_1\right)}{e\left(\prod_{j\in S} g_{n+1-j}, c_2\right)}, c_1, c_3. \quad (5)$$

For each $i \in S$ (3), subsequently the user could check whether the return files are correct when receive the proofs (4).

However, if the cloud forges the proofs, we get

$$\text{proof}_i = \begin{cases} (p_1, p_2, p_3), & i \in (S - I'), \\ (p_1, p_2, \overline{p_3}), & i \in I', \end{cases} \quad (6)$$

and returns the result $(I - I')$ to the user, which can also pass the user's verify algorithm, where $\overline{p_3}$ is a fake value as long as it is not equal to p_3, and $I' = \{f_1, f_2, \ldots, f_k\} (k \le t)$ is the file set lost by the cloud.

For each $i \in I'$, the user computes

$$M' = p_1 \cdot e(K_a, p_2), \overline{BF_i} = H_1(M') \oplus \overline{p_3}, \quad (7)$$

and verifies

$$BF\text{verify}(\{H'_1, \ldots, H'_k\}, \overline{BF_i}, w) \neq 1. \quad (8)$$

Because $\overline{p_3}$ is forged by the cloud server, and $\overline{BF_i}$ will be an incorrect value, so (4) would not hold true. While for each $i \in (I - I')$, p_3 is correct, so (4) holds true. Thus, the user believes that the search result $(I - I')$ is complete.

Therefore, Liu et al.'s scheme cannot correctly verify the completeness of search results.

5. System Model and Definitions

In this section, we first present the model and definition, then describe the requirements.

5.1. System Model. The system model is shown in Figure 2, which contains four entities: publisher, cloud server, edge server, and data user.

(i) *Publisher.* When the publisher has the shared data, it encrypts the data and outsources to the cloud. After the purchase action, the publisher distributes the corresponding keys to user and edge for the data query and verification.

(ii) *Cloud Server.* The cloud stores the uploading data and responds on the query. The cloud is not fully trustworthy.

(iii) *Edge Server.* The edge assists users with the query and verification operation. It is trusted and like a front-end server on the user side.

(iv) *Data User.* The users purchase data and obtain the search authorization, who can query data with the assistance of edge server.

5.2. Formal Definition

(1) Init $(1^\lambda, n) \longrightarrow PP$: This algorithm is operated by the publisher. The algorithm inputs a security parameter λ and the maximum file number n and then outputs the system public parameters PP

(2) KeyGen $(PP) \longrightarrow (E, sk, PK)$: This algorithm is run by the publisher. The algorithm inputs the parameters PP and then outputs hash key E, publisher's secret key sk, and public key PK

(3) Encrypt $(PP, PK, W) \longrightarrow (C, BF, L_{\text{root}})$: This algorithm is run by the publisher, and it inputs parameters PP, public key PK, and keyword set W and then outputs the ciphertext set C, Bloom filter value set BF, and the root node value of a Merkle hash tree L_{root}

(4) Authorization $(PP, sk, id, S) \longrightarrow (A_{id}, K_{id})$: This algorithm is run by the publisher. The algorithm inputs parameters PP, secret key sk, user's public identity id, and a subset $S(S \subseteq N)$ and then outputs the authorization key A_{id} and identity key K_{id}

(5) Trapdoor $(E, A_{id}, w, K_{id}) \longrightarrow \text{Tr}$: The algorithm is operated by the data user and edge server. It takes the authorization key A_{id}, hash key E, keyword w, and identity key K_{id} as the input and then outputs the trapdoor Tr

(6) Search $(PP, C, BF, \text{Tr}) \longrightarrow (I, PF(S))$: This algorithm inputs parameters PP, the ciphertext set C, Bloom filter value set BF, and trapdoor Tr and then outputs the search result I and verification proof $PF(S)$

(7) Verification $(PP, I, PF(S), BF, L_{\text{root}}) \longrightarrow (1/0)$: This algorithm takes public parameters PP, search result I, verification proof $PF(S)$, and publisher's public key (BF, L_{root}) as the input and then outputs 1 if the result is correct and complete; otherwise, it outputs 0

Definition 1. Our proposed scheme includes the below seven algorithms.

5.3. Requirements

5.3.1. Security. In our proposed scheme, the system should satisfy indistinguishability against selective-file chosen keyword attack (IND-SF-CKA) security [30].

We used a game between a Challenger Cha and a polynomial-time Adversary Ad to define the model of IND-SF-CKA.

(1) *Initial.* Ad publishes the file set S^* to be attacked.

(2) *Setup.* Cha builds the system and generates the public parameter and then sends the parameters and keyword space W to Ad.

(3) *Process 1.* Ad carries out a series of Authorized and Trapdoor queries as follows:

Authorized query: Ad sends any file set S to Cha which cannot have any intersection with S^* and receives the keys (A_{id}, K_{id}) computed by Cha through the algorithm Authorize (PP, sk, id, S).

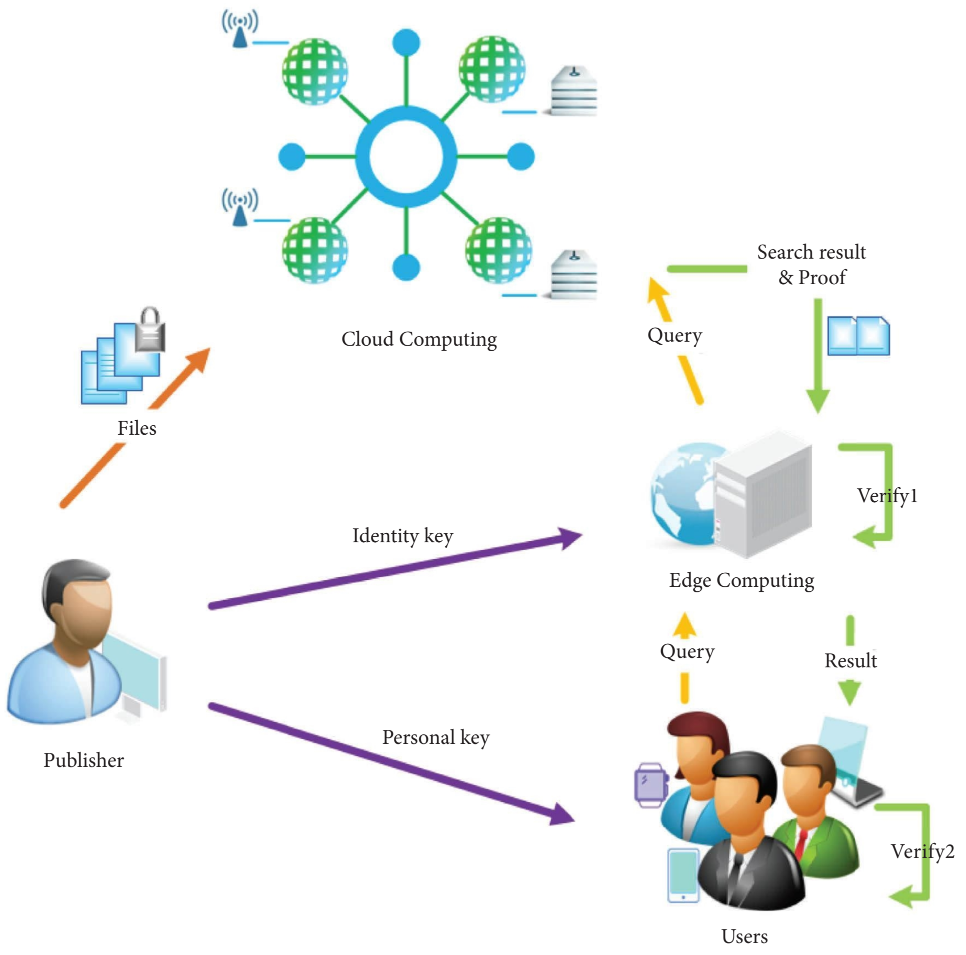


FIGURE 2: System model.

Trapdoor query: Ad can adaptively enquire to Cha with any keyword $w \in W$ in S and then Cha runs the algorithm Trapdoor (E, A_{id}, w, K_{id}) to generate the trapdoor Tr and sends it to Ad.

(4) *Challenge*. When Ad wants to finish Process 1, it generates two keywords of the same length in plain text $(w_0, w_1) \in W$. Cha randomly selects $u \in \{0, 1\}$ and $i \in S^*$ and then operates the Encrypt algorithm to output the challenge ciphertext $cw^* =$ Encrypt (PP, PK, w_b) and sends cw^* to Ad.

(5) *Process 2*. Ad continuously sends the queries of Authorize and Trapdoor algorithms as in Process 1, the limit is the Authorize queries of S cannot have any intersection with S^*, and the Trapdoor query of w_0 or w_1 cannot be in S^*.

(6) Guess. Ad guesses the value of u and outputs u'. Ad wins the game if $u' = u$. Set that Ad's advantages over this game is

$$\mathrm{Adv}_{\mathrm{Ad}}^{\mathrm{scheme}} = \left|\Pr\left[u' = u\right] - \frac{1}{2}\right|. \tag{9}$$

Definition 2. The proposed scheme is IND-SF-CKA secure if $Adv_{Ad}^{\mathrm{scheme}}$ is negligible.

5.3.2. Consistency. Besides the security, the proposed scheme should satisfy the consistency when using the trapdoor for query [36].

$$\Pr\left[\mathrm{Search}\left(\mathrm{Trapdoor}\left(E, A_{id}, w', K_{id}\right), \mathrm{Encrypt}\left(PP, PK, w\right)\right) = 1\right] = 0. \tag{10}$$

Definition 3. For all distinct keywords $w, w\prime \in \{0, 1\}^*$, the trapdoor with w' and the cipher text for encrypting w is consistent if and only if the following probability holds true.

T algorithm Search is always rejected when the keyword $w\prime$ contained in Trapdoor is different from the keyword w in Encrypt.

5.3.3. Correctness and Completeness. The proposed scheme should satisfy that the search result is correct and complete, which is defined as follows based on the definition in [37].

(1) Correctness: $w \subset f_1 \wedge w \subset f_2 \wedge \ldots \wedge w \subset f_t$

(2) Completeness: $(f_1 - Q) \cap (f_2 - Q) \cap \ldots \cap (f_s - Q) = \varnothing$, where $Q = f_1 \cap f_2 \cap \ldots \cap f_t$

Definition 4. For authorized file set $S = \{1, 2, , \ldots, s\}$ and a keyword w for the query, the search result with $I = \{f_1, f_2, \ldots, f_t\}$ is correct and complete when the below two terms are true.

The correctness condition ensures that the search result contains keyword w, and the completeness condition guarantees the search result which includes all files containing keyword w. When the search result is correct and complete, the Verification algorithm would output 1; otherwise, it would output 0.

5.4. The Proposed Scheme

5.4.1. Overview. In scheme [15], the fundamental reason why the cloud can launch forgery attacks is that the verification process requires cloud to participate in calculation, and the flow is "publisher computing - > cloud computing - > user computing, so the users will not know whether the cloud has forged proof. In our design, we changed the verification flow to "publisher computing - > public proof - > user computing," And the cloud does not participate in computing but only forwards the data. Besides, in the phase of "public proof," we use the Merkle hash tree to ensure that the proof is not tampered with by the cloud, thus guaranteeing the correctness and completeness of search data.

Secondly, in the index encryption phase of existing researches [13–15, 29–31], they encrypt every keyword associated with each file and then upload them. Consequently, the search function needs to match whether the query value is equal to each keyword, which will greatly affect the search efficiency. In our design, we store each file's keyword set in a Bloom filter; that is, only one string is uploaded for one file. Hence, during the keyword processing, the communication and storage cost is greatly reduced, and the overhead is related to the number of files but not to the number of keywords.

Thirdly, in the existing works [13–15, 32, 33], the verification process are mostly based on the cloud computing environment, and users need to perform generous calculations to verify, which is unacceptable for some resource-constrained clients. In this paper, the edge server is introduced to assist users in search and verification, which greatly reduces the computation overhead for users and make them more comfortable.

5.4.2. Construction. Based on the definition described in 5.2, we propose a concrete construction as follows:

(1) Init $(1^\lambda, n) \longrightarrow PP$: The publisher generates the public parameters as given below.

Generates a bilinear map system $\mathbb{B} = (p, \mathbb{G}, \mathbb{G}_1, e(\cdot,\cdot))$, where p is the order of $\mathbb{G}$ and $2^\lambda \leq p \leq 2^{\lambda+1}$.
Sets n as the maximum number of files and then the complete file index set $N = \{1, \ldots, n\}$.
Picks a random generator $\vartheta \in \mathbb{G}$ and a random number $a \in \mathbb{Z}_p$ and computes $\vartheta_i = \vartheta^{a^i}$, for $i = (1, 2, \ldots, n, n+2, \ldots, 2n)$.
Chooses a collision-free hash function used for the Merkle hash tree: H_0: $\{0,1\}^* \longrightarrow \mathbb{G}$.
Chooses l as the maximum length of Bloom filter, and k independent universal hash functions: $H_1, \ldots, H_k$: $\{0,1\}^* \longrightarrow \{0,1\}^l$.
Chooses a keyed hash function H: $\{0,1\}^* \longrightarrow \mathbb{G}$.

Then the system public parameters are

$$PP = (\mathbb{B}, \{\vartheta, \vartheta_1, \ldots, \vartheta_n, \vartheta_{n+2}, \ldots, \vartheta_{2n}\}, H_0, \{H_1, \ldots, H_k\}, H)_{\text{key}}. \quad (11)$$

(2) KeyGen $(PP) \longrightarrow (E, sk, PK)$: The publisher generates a random string E as the hash key and chooses a random number $\gamma \in \mathbb{Z}_p$ as the secret key, then the public key is

$$PK = v = \vartheta^\gamma. \quad (12)$$

(3) Encrypt $(PP, PK, W) \longrightarrow (C, BF, L_{\text{root}})$: The publisher encrypts the files and keywords as follows:

Firstly, for each file index $i \in N$, we get those as follows:

Generates an empty Bloom filter BF_i = BFInit.
Randomly chooses an integer $t \in \mathbb{Z}_p$ and computes two encryption auxiliary values:

$$\begin{aligned} c_{i,1} &= \vartheta^t, \\ c_{i,2} &= (v \cdot \vartheta_i)^t. \end{aligned} \quad (13)$$

For the keywords $\{w_{ij}\}$ $(j \in \{1, \ldots, x\})$ contained by the i-th file, we get

Then set $BF = \{BF_i\}, i \in N$.

Secondly, the publisher generates a MHT with n leaf nodes to guarantee the correctness of the file auxiliary values. Then the MHT is built as follows:

Each leaf node's value is a hash of two auxiliary values such as

$$L_i = H_0(c_{i,1} c_{i,2}), i \in N. \quad (14)$$

Each nonleaf node's value is a hash of its two children nodes and then the root node's value L_{root} can also be determined.

Finally, the publisher uploads $C = \{c_{i,1}, c_{i,2}\}$ $(i \in N)$ to the cloud and adds BF and L_{root} to his/her public key as

$$PK = (v, BF, L_{\text{root}}). \quad (15)$$

Note that the cloud can rebuilt the MHT since it has the encryption auxiliary values C.

(4) Authorization $(PP, sk, id, S) \longrightarrow (A_{id}, K_{id})$: For user's public identity id and authorization file subset $S \subseteq N$, the publisher computes as

$$A_{\mathrm{id}} = \mathrm{id}^{\gamma},$$
$$K_{\mathrm{id}} = \prod_{j \in S} \frac{\vartheta_{n+1-j}^{\gamma}}{A_{id}}, \tag{16}$$

and then sends the authorization key (E, A_{id}) to user and identity key K_{id} to edge.

(5) Trapdoor $(E, A_{id}, w, K_{id}) \longrightarrow \mathrm{Tr}$: For the keyword w, the user generates the search query Q and sends to edge server as

$$Q = H_E(w) \cdot A_{id}. \tag{17}$$

When the search query is recieved, the edge first verifies the user's public identity, then uses the corresponding identity key to compute the trapdoor, and sends to cloud:

$$\mathrm{Tr} = K_{id} \cdot Q. \tag{18}$$

(6) Search $(PP, C, BF, \mathrm{Tr}) \longrightarrow (I, PF(S))$: On receiving Tr, the cloud server does the following works.

Firstly, for each file index $i \in S$:

Computes the corresponding ciphertext for the i-th file as

$$cw_i = \frac{e\left(\mathrm{Tr} \cdot \prod_{j \in S, j \neq i} \vartheta_{n+1-j+i}, c_{1,i}\right)}{e\left(\prod_{j \in S} \vartheta_{n+1-j}, c_{2,i}\right)}. \tag{19}$$

Gets the matching result of the i-th file through the Bloom filter as

If $\mathrm{test}_i = 1$, the file f_i is added to the result set I.

Secondly, the cloud server constructs the verification proof $PF(S)$ as follows:

Auxiliary values $C = \left\{c_{i,1}, c_{i,2}\right\}, i \in S$.
Merkle hash tree proof is denoted as $pf(S)$:

where $\mathrm{path}(i)$ is a list of nodes that denotes the path from leaf node i to the root node of Merkle hash tree. Let node_{N-S} be a list of leaf nodes excluding subset S. If there are sibling nodes in node_{N-S}, then the sibling nodes are replaced by their parent nodes, which is denoted as $\mathrm{node}_{N-S}^{\min}$. While $L_{\mathrm{node}_{N-S}^{\min}}$ is the list containing the hashes of the nodes in $\mathrm{node}_{N-S}^{\min}$, which can calculate L_{root} with the hashes of the nodes in set S, then the verification proof is

$$PF(S) = (C, pf(S)). \tag{20}$$

Finally, the cloud sends the search result and proof $(I, PF(S))$ to the edge.

Take the case of $n = 8$ as an example, as shown in Figure 3. Assume $S = \{1, 2, 8\}$ and search result $I = \{f_2, f_8\}$, then the verification proof is generated as follows:

Auxiliary values: $C = \left\{(c_{1,1}, c_{1,2}), (c_{2,1}, c_{2,2}), (c_{8,1}, c_{8,2})\right\}$.
The $\mathrm{path}(i)$ from the node i to the root node is generated such that

Leaf node $i = 1$, $\mathrm{path}(i = 1) = (1, 9, 13, \mathrm{root})$
Leaf node $i = 2$, $\mathrm{path}(i = 2) = (2, 9, 13, \mathrm{root})$
Leaf node $i = 8$, $\mathrm{path}(i = 8) = (8, 12, 14, \mathrm{root})$

Then $\mathrm{node}_{N-S}^{\min}$ is computed as

$$\begin{aligned}\mathrm{node}_{N-S}^{\min} &= \mathrm{node}_{\{n_1,\ldots,n_8\}-\{n_1,n_2,n_8\}}^{\min} \\ &= \mathrm{node}_{\{n_3,n_4,n_5,n_6,n_7\}}^{\min} = \{n_{10}, n_{11}, n_7\}.\end{aligned} \tag{21}$$

Finally, we get

$$\begin{aligned}pf(S) &= \left(\{\mathrm{path}(i)\}, i \in S, L, \mathrm{node}_{N-S}^{\min}\right) \\ &= \left(\{\mathrm{path}(1), \mathrm{path}(2), \mathrm{path}(8)\}, \{L_{10}, L_{11}, L_7\}\right).\end{aligned} \tag{22}$$

(7) Verification $(PP, I, PF(S), BF, L_{\mathrm{root}}) \longrightarrow (1/0)$: When the search proofs are recieved, the edge server does the following work.

Firstly, for each file index $i \in S$, the edge computes $L_{i'} = H_0(\|c_{i,1}c_{i,2}\|)$, recovers L_{root}' based on $pf(S)$, and then verifies if $L_{\mathrm{root}}' = L_{\mathrm{root}}$. If holds, then C is correct. Otherwise, it is $\perp$.

Secondly, for $i \in S$, the edge computes the auxiliary verification value as

$$cw_i = \frac{e\left(\mathrm{Tr} \cdot \prod_{j \in S, j \neq i} \vartheta_{n+1-j+i}, c_{1,i}\right)}{e\left(\prod_{j \in S} \vartheta_{n+1-j}, c_{2,i}\right)}, i \in S. \tag{23}$$

Finally, the edge sends the search result I and auxiliary verification value $CW = \{cw_i\}, i \in S$ to user.

After receives the auxiliary verification value, the user can verify the search result:

$$\mathrm{verify}_i = BF\mathrm{Query}\left(\{H_1, \ldots, H_k\}, BF_i, cw_i\right), i \in S. \tag{24}$$

If for each $i \in I$, $\mathrm{verify}_i = 1$, and for each $i \in (S - I)$, $\mathrm{verify}_i = 0$, and then it outputs 1, which means the search result is correct and complete. Otherwise, it outputs 0.

6. Requirements Analysis

6.1. Security. We will prove that our scheme is IND-SF-CKA secure under the standard model based on the Decision l-BDHE assumption. The security is verified by the below theorem.

Theorem 1. *If the decision l-BDHE problem is hard to solve, then our proposed scheme satisfies the IND-SF-CKA security.*

Proof. Suppose there is an Adversary Ad which can destroy the IND-SF-CKA security of the proposed scheme, then a Challenger Cha can build an Algorithm D to solve the decision l-BDHE problem. This contradicts our hypothesis

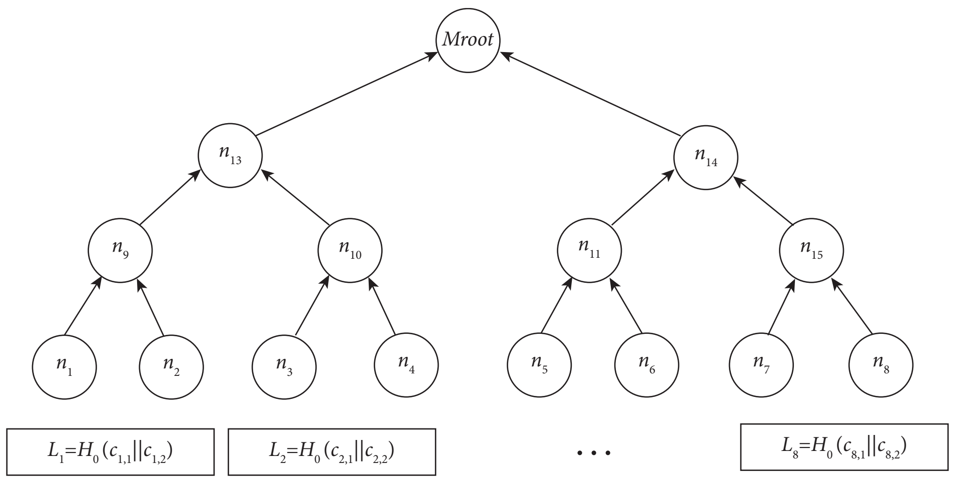


Figure 3: Merkle hash tree.

that the decision l-BDHE problem is hard to solve; thus, it is proved that the scheme is IND-SF-CKA secure.

Suppose D is given as an instance of decision l-BDHE problem, which includes a bilinear pairing system e: $\mathbb{B} = (p, \mathbb{G}, \mathbb{G}_1, e(\cdot,\cdot))$ and parameters $\vartheta_i = \vartheta^{a^i}$, for $i = (1, 2, \ldots, l, l+2, \ldots, 2l)$, where a is unknown. Then D is further given as $h \in \mathbb{G}$, $Z \in \mathbb{G}_1$. At last, the input is $(h, \vartheta, \vartheta^a, \vartheta^{a^2}, \ldots, \vartheta^{a^l}, \vartheta^{a^{l+2}}, \ldots, \vartheta^{a^{2l}}, Z)$, and D needs to decide if $Z = e(\vartheta^{a^{l+1}}, h)$ or Z is a random value in $\mathbb{G}_1$.

D simulates the Challenger Cha to begin a game with Ad as follows:

(1) Initial: D receives the file set S^* that Ad wants to be challenged. Then D gets the parameters $(h, \vartheta, \vartheta^a, \vartheta^{a^2}, \ldots, \vartheta^{a^l}, \vartheta^{a^{l+2}}, \ldots, \vartheta^{a^{2l}}, Z)$ of the decision l-BDHE problem.

(2) Setup: D generates some other parameters and then provides them with the given parameters to Ad as follows:

(1) Chooses a hash function H: $\{0,1\}^* \longrightarrow \mathbb{G}$.

(2) Sets W as the keyword space.

(3) Randomly chooses $r \in \mathbb{Z}_p$, a file index $i \in S^*$, and sets $PK = \vartheta^r/\vartheta_i$, where ϑ_i is from the given parameters ϑ^{a^i}, which is corresponding to the chosen class i, then the sk is $r - a^i$ which is unknown.
Then D sends $(\mathbb{B}, \vartheta, \vartheta^a, \vartheta^{a^2}, \ldots, \vartheta^{a^l}, \vartheta^{a^{l+2}}, \ldots, \vartheta^{a^{2l}}, HW, PK)$ to Ad.

(3) Process 1: Ad carries out a series of adaptive queries:

Authorize query: Ad can query any file set S, where $S \cap S^* = 0$. Then D computes

$$A_{id} = id^{\gamma},$$

$$K_{id} = \prod_{j \in S} \vartheta_{n+1-j}^{\gamma - a^i} = \prod_{j \in S} \frac{\left(\vartheta_{n+1-j}^{\gamma}/\vartheta_{n+1-j+i}\right)}{A_{id}}, \tag{25}$$

and sends $(A_{\text{id}}, K_{\text{id}})$ back.

Trapdoor query: Ad can query any file set S and any keyword w, where $S \cap S^* = 0$, $w \in W$. Then D generates $\text{Tr} = A_{\text{id}} \cdot H_E(w) \cdot K_{\text{id}}$, and sends Tr to Ad.

(4) Challenge: When Ad determines to finish Process 1, it sends two equal length keywords $(w_0, w_1) \in W$ to D, which cannot be the same as the Trapdoor query of Process 1 before. Then D performs the following operations:

(1) Randomly picks a coin $u \in \{0,1\}$.

(2) Sets $h = \vartheta^t$, where t is unknown, thus

Here, $PK = \vartheta^r/\vartheta_i$, then $C^* = (c_1 = h, c_2 = h^r, cw = e(H_E(w_u), h)/Z)$ is sent to Ad. Notice that if $Z = e(\vartheta^{a^{l+1}}, h)$, then $cw = e(H_E(w_b), h)/e(\vartheta^{a^{l+1}}, h)$, and C^* is a valid ciphertext of encrypting the keyword w_u with the class i. Otherwise, Z is a random element in $\mathbb{G}_1$.

(5) Process 2: Similar to Process 1, Ad continues to conduct Authorize and Trapdoor queries, and D adopts the same strategy to reply.

(6) Guess: Ad guesses the value of u and outputs u'. If $u' = u$, then D outputs 1, which means $Z = e(\vartheta^{a^{l+1}}, h)$. Otherwise, the output is 0 implies Z is a random value in $\mathbb{G}_1$.

(7) Probability analysis: If $Z = e(\vartheta^{a^{l+1}}, h)$, then C^* is valid. Otherwise, Z is a random value in $\mathbb{G}_1$, and C^* is an invalid ciphertext. In such case, the advantage of Ad is equal to 1/2. If Ad successfully implements IND-SF-CKA to the scheme by the advantage of ε as $\Pr[u' = u - 1/2] \geq \varepsilon$, then D can solve the decision l-BDHE problem by the advantage $\text{ADV}_D^{\text{Decisionl-BDHE}} = |\Pr[D(h, \vartheta, \vartheta^a, \vartheta^{a^2}, \ldots, \vartheta^{a^l}, \vartheta^{a^{l+2}}, \ldots, \vartheta^{a^{2l}}), Z, 0] - \Pr[D(h, \vartheta, \vartheta^a, \vartheta^{a^2}, \ldots, \vartheta^{a^l}, \vartheta^{a^{l+2}}, \ldots, \vartheta^{a^{2l}}, Z \neq e(\vartheta, h)^{a^{l+1}}) = 0] \quad | \geq \varepsilon = e(\vartheta, h)^{a^{l+1}})$. However, this is contrary to Definition 2, so Ad cannot implement IND-SF-CKA to the scheme by the advantage of ε.

Therefore, the proposed scheme satisfies the IND-SF-CKA security.

6.2. Consistency. Proof.If H_{key} is collision-free, then there must be different keywords $(w, w\prime) \in W$ so that $H_E(w) \neq H_E(w\prime)$. According to equations (3)–(5), we can get

$$\begin{aligned} cw_i &= \frac{e\left(\mathrm{Tr} \cdot \prod_{j\in S, j\neq i} \vartheta_{n+1-j+i}, c_{1,i}\right)}{e\left(\prod_{j\in S} \vartheta_{n+1-j}, c_{2,i}\right)}, \\ &= \frac{e\left(A_{id} \cdot H_E(w) \cdot K_{id} \cdot \prod_{j\in S, j\neq i} \vartheta_{n+1-j+i}, \vartheta^t\right)}{e\left(\prod_{j\in S} \vartheta_{n+1-j}, (v \cdot \vartheta_i)^t\right)}, \\ &= \frac{e\left(\prod_{j\in S} \vartheta^{\gamma}_{n+1-j} \cdot H_E(w) \cdot \prod_{j\in S, j\neq i} \vartheta_{n+1-j+i}, \vartheta^t\right)}{e\left(\prod_{j\in S} \vartheta_{n+1-j}, v^t\right) \cdot e\left(\prod_{j\in S} \vartheta_{n+1-j}, \vartheta^t_i\right)}, \\ &= \frac{e\left(\prod_{j\in S} \vartheta^{\gamma}_{n+1-j}, \vartheta^t\right) e\left(\prod_{j\in S, j\neq i} \vartheta_{n+1-j+i}, \vartheta^t\right) e\left(H_E(w), \vartheta^t\right)}{e\left(\prod_{j\in S} \vartheta_{n+1-j}, v^t\right) \cdot e\left(\prod_{j\in S} \vartheta_{n+1-j}, \vartheta^t_i\right)}, \\ &= \frac{e\left(H_E(w), \vartheta\right)^t}{e\left(\vartheta_1, \vartheta_n\right)^t}. \end{aligned} \tag{26}$$

Theorem 2. *If H_{key} is a collision-free hash function, then the proposed scheme is consistent.*

Then for the same keyword, the cloud uses Trapdoor Tr to generate the matching ciphertext cw_i, which is equal to the encrypted keyword generated by the publisher, then

$$\Pr\left[\mathrm{Search}\begin{pmatrix} \mathrm{Trapdoor}\left(E, A_{id}, w, K_{id}\right), \\ \mathrm{Encrypt}\left(PP, PK, w\right) \end{pmatrix} = 1\right] = 0. \tag{27}$$

Therefore, the proposed scheme is consistent.

6.3. Correctness and Completeness

Theorem 3. *If $(H_1, \ldots, H_k)$ are random and the Bloom filter has a low false-positive rate, then the user can verify whether the result is correct and complete.*

Proof. Correctness: In Theorem 2, our scheme is proved to be consistent. Then (27) can be used to ensure the return files are correct. Suppose the Bloom filter has a low false-positive rate, when (27) is equal to 1, it means the i-th file contains the keyword w: $w \subset f_i$.

Completeness: In our scheme, for the file set $S = \{f_1, f_2, \ldots, f_s\}$, and given a query keyword w from a user, the cloud returns the search result $I = \{f_1, f_2, \ldots, f_t\}$, $(t \leq s)$. With equations (7) and (8), if for each $i \in I$, it satisfies $verify_i = 1$, and for each $i \in (S - I)$, it satisfies $verify_i = 0$, then $(f_1 - Q) \cap (f_2 - Q) \cap \ldots \cap (f_s - Q) = \varnothing$, where $Q = f_1 \cap f_2 \cap \ldots \cap f_t$, so that the search result is complete; that is, the cloud server returns all files that include the keyword w and does not return the files that do not include the keyword w. Otherwise, it is incomplete. Meanwhile, the correction of the encryption auxiliary value is guaranteed by the Merkle hash tree, which the cloud cannot forge.

Therefore, the user can verify whether the result is correct and complete.

7. Performance

7.1. Performance Analysis. In this section, we compare the other two related schemes [14, 15, 29] in terms of various costs so as to analyze the performance. We define the below notations in Table 1.

7.1.1. Functionality. We show the function comparison in Table 2, and we can see that all schemes realize the function of fine-grained authorization. In the verification function, scheme [14] can only verify the correctness, and scheme [15] can verify the correctness and completeness but exists security defect, and scheme [29] does not consider the verification module, while our scheme implements all the verification in the cloud-edge environment.

7.1.2. Computation Cost. The comparison results of computation cost for several schemes are shown in Table 3, and it demonstrates that our scheme has less computation overhead than schemes [14, 15] on the whole.

In the phases of Init, KeyGen, Extract, and Trapdoor, all four schemes have the same efficiency, which is because the parameters and data generated are alike. Note that the hash computation overhead is $O(1)$, which is far more cheaper than others, so we did not include it to the statistic.

In the Encrypt phase, our scheme generates $2n$ parameters and nr ciphertexts, which is the same as scheme [29]. While scheme [14] needs to generate $3n$ parameters and nr ciphertexts, and scheme [15] generates $4n$ parameters and nr ciphertexts, so the Encrypt process overhead of our scheme is less than schemes [14, 15].

In the Search phase, the overhead of our scheme is the same as scheme [29] and is less than schemes [14, 15].

In the Verification phases, the user overhead of our scheme is less than schemes [14, 15], which is because the edge server undertakes most calculations.

Then, the total computational cost of our scheme is less than schemes [14, 15].

7.1.3. Communication Cost. Table 4 shows the communication overhead comparison as it can be seen that our scheme has less communication overhead than other schemes overall.

In the Encrypt phase, our scheme has less cost than other schemes and that is because our scheme uploads only one Bloom filter value for one file, and other schemes need to upload all the encrypted keywords. Then the communication cost of our scheme for keyword cipher text processing is related to the number of files but not to the number of

Table 1: Notations.

Notation	Meaning
$Mul_{\mathbb{G}}$	Multiplication manipulation in $\mathbb{G}$ or $\mathbb{G}_1$
$Ex_{\mathbb{G}}$	Exponentiation manipulation in $\mathbb{G}$ or $\mathbb{G}_1$
$Ex_{\mathbb{Z}_p}$	Exponentiation manipulation in $\mathbb{Z}_p$
pair	Pairing operation
H	Hash operation
n	Total file number
r	The average number of keywords per file
l	The length of an element in Bloom filter
m	The length of Merkle hash tree node
p	The size of an element in $\mathbb{G}$
q	The size of an element in $\mathbb{G}_1$

Table 2: Functionality comparison.

Scheme	Fine-grained authorization	Completeness	Environment
Scheme [14]	Yes	Yes/no	Cloud
Scheme [15]	Yes	Security defect	Cloud
Scheme [29]	Yes	No/no	Cloud
Our scheme	Yes	Yes/yes	Cloud-edge

Table 3: Computation cost comparison.

Scheme	Init	KeyGen
Scheme [14]	$2n(Ex_{\mathbb{Z}_p} + Ex_{\mathbb{G}})$	$Ex_{\mathbb{G}}$
Scheme [15]	$2n(Ex_{\mathbb{Z}_p} + Ex_{\mathbb{G}})$	$Ex_{\mathbb{G}}$
Scheme [29]	$2n(Ex_{\mathbb{Z}_p} + Ex_{\mathbb{G}})$	$Ex_{\mathbb{G}}$
Our scheme	$2n(Ex_{\mathbb{Z}_p} + Ex_{\mathbb{G}})$	$Ex_{\mathbb{G}}$
Scheme	Authorization	Encrypt
Scheme [14]	$nEx_{\mathbb{G}}$	$4nr \cdot Ex_{\mathbb{G}} + 3n \cdot Mul_{\mathbb{G}} + 2nr \cdot \text{pair}$
Scheme [15]	$nEx_{\mathbb{G}}$	$(3n + nr)Ex_{\mathbb{G}} + (2nr + n)\text{pair}$
Scheme [29]	$nEx_{\mathbb{G}}$	$(2n + nr)Ex_{\mathbb{G}} + 2nr \cdot \text{pair}$
Our scheme	$(n + 1)Ex_{\mathbb{G}}$	$(2n + nr)\mathrm{Ex}_{\mathbb{G}} + 2nr \cdot \text{pair}$
Scheme	Trapdoor	Search
Scheme [14]	Mul_G	$(2n + 1)\mathrm{Mul}_{\mathbb{G}} + Ex_{\mathbb{G}} + 4n \cdot \text{pair}$
Scheme [15]	Mul_G	$6n \cdot Mul_{\mathbb{G}} + 4n \cdot \text{pair}$
Scheme [29]	Mul_G	$3n \cdot \mathrm{Mul}_{\mathbb{G}} + 2n \cdot \text{pair}$
Our scheme	Mul_G	$3n \cdot \mathrm{Mul}_{\mathbb{G}} + 2n \cdot \text{pair}$
Scheme	$\text{Verification}_{\text{user}}$	$\text{Verification}_{\text{user}}$
Scheme [14]	$n \cdot \mathrm{Mul}_{\mathbb{G}} + n \cdot \text{pair}$	
Scheme [15]	$n \cdot \mathrm{Mul}_{\mathbb{G}} + n \cdot \text{pair}$	—
Scheme [29]	—	—
Our scheme	$n \cdot H$	$3n \cdot \mathrm{Mul}_{\mathbb{G}} + 2n \cdot \text{pair}$

Table 4: Communication cost comparison.

Scheme	Encrypt	Trapdoor	Search
Scheme [14]	$2np + nrq + nrp$	p	$n + np + nrq$
Scheme [15]	$2np + nq + nrq + nl$	p	$n + np + nq + nl$
Scheme [29]	$2np + nrq$	p	n
Our scheme	$2np + nl$	p	$n + 2np + \sqrt{nm}$

keywords. Note that the number of files is n and the number of keywords in each file is k, then our scheme only needs to pass n Bloom filter values and $2n$ elements in $\mathbb{G}$ to the cloud. While scheme [29] needs to transmit $2n$ elements in $\mathbb{G}$ and nr elements in $\mathbb{G}_1$, scheme [14] needs to transmit $2n + nr$ elements in $\mathbb{G}$ and nr elements in $\mathbb{G}_1$, and scheme [15] has the the biggest overhead which needs to transmit $2n$ elements in $\mathbb{G}$, $(n + nr)$ elements in $\mathbb{G}_1$, and n elements of Bloom filter.

In the Trapdoor phase, all schemes have the same communication overhead.

In the Search phase, all four schemes need to return n -bits search results, and scheme [14] still needs to transmit two auxiliary values $(np + nrq)$ for verification, and scheme [15] still needs to transmit three auxiliary values $(np + nq + nl)$ for verification, while our scheme needs two auxiliary values $(2np + \sqrt{nm})$.

Since the scheme [29] has no verification function, the communication cost of our scheme is higher than scheme [29] but is less than schemes [14, 15].

7.1.4. Storage Cost. We display the storage cost comparison in Table 5, which demonstrates that our scheme has less storage overhead than other two schemes in general.

In the publisher/data owner side, all schemes store one key, and then they have the same cost with p.

In the user side, our scheme stores two keys, and other schemes only need to store one key.

In the cloud side, our scheme has less storage cost than other three schemes. In our scheme, the cloud server only needs to store one Bloom filter value for one file, while other two schemes need to store all the encrypted keywords. Therefore, the cloud's storage cost in our scheme is only related to the number of files and others are related to the number of keywords. Then our scheme needs $2n$ elements in $\mathbb{G}$ and n elements of Bloom filter to store, while scheme [29] stores $2n$ elements in $\mathbb{G}$ and nr elements in $\mathbb{G}_1$, scheme [14] stores $(2n + nr)$ elements in $\mathbb{G}$ and nr elements in $\mathbb{G}_1$, and scheme [15] stores $2n$ elements in $\mathbb{G}$, $(n + nr)$ elements in $\mathbb{G}_1$, and n elements of Bloom filter.

Therefore, our scheme has less storage cost than other three schemes.

7.2. Experimental Results. To assess the actual performance, we carry out the comparison experiments with schemes [14, 15, 29]. We use the latest JPBC library to deploy the archetypes of these schemes. Our experiments run on a computer with Intel Core i5-6550 CPU working in theWindows 7 system. We used the Enron database as the source, which extracted 10 thousands documents, 6 thousands keywords, and 80 thousands keyword-document pairs, of which most files matched fewer than 10 keywords. We selected the Type-A pairing to complete the specific algorithm, and then experimented with different number of files and keywords within four schemes. The experiment results are consistent with our performance analysis, as shown in Figure 4.

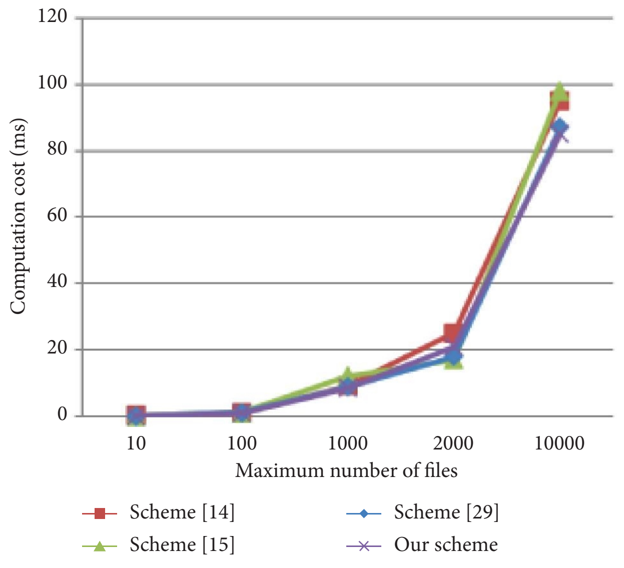
In the Init and Authorization phases, all schemes have the same computation cost, which is liner increasing with the number of files, as shown in Figures 4(a) and 4(c).

In the KeyGen and Trapdoor phases, all schemes have the same operation time for different numbers of files, as shown in Figures 4(b) and 4(e).

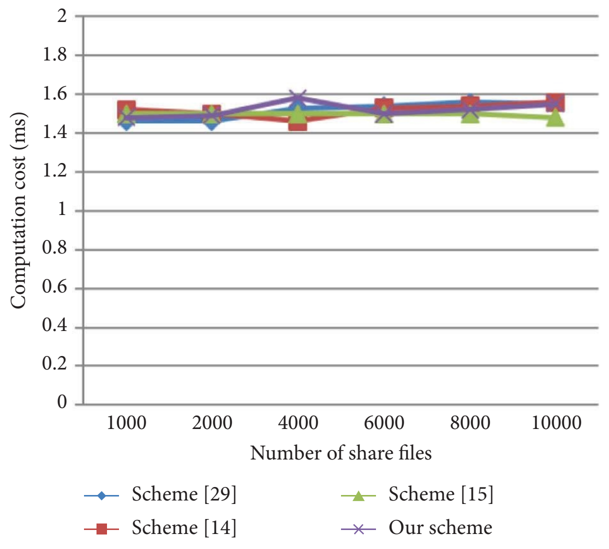
In the the Encrypt and Search phases, our scheme has the same operation time as scheme [29], which is less than schemes [14, 15], as shown in Figures 4(d) and 4(f).

Table 5: Storage cost comparison.

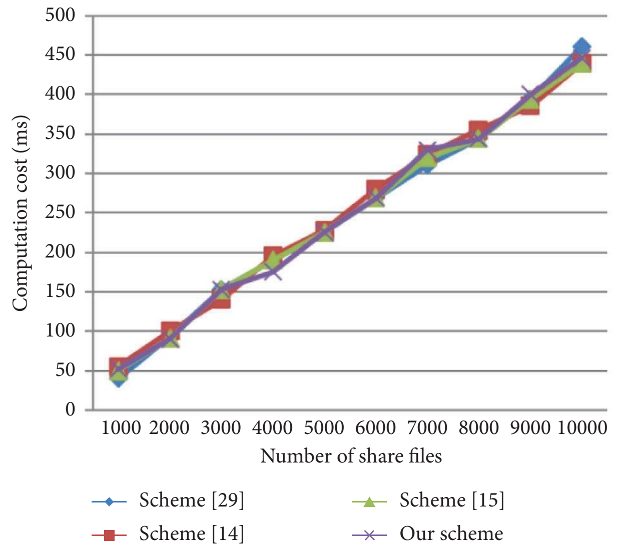
Scheme	Publisher	User	Cloud
Scheme [14]	p	p	$2np + nrq + nrp$
Scheme [15]	p	p	$2np + nq + nrq + nl$
Scheme [29]	p	p	$2np + nrq$
Our scheme	p	$2p$	$2np + nl$



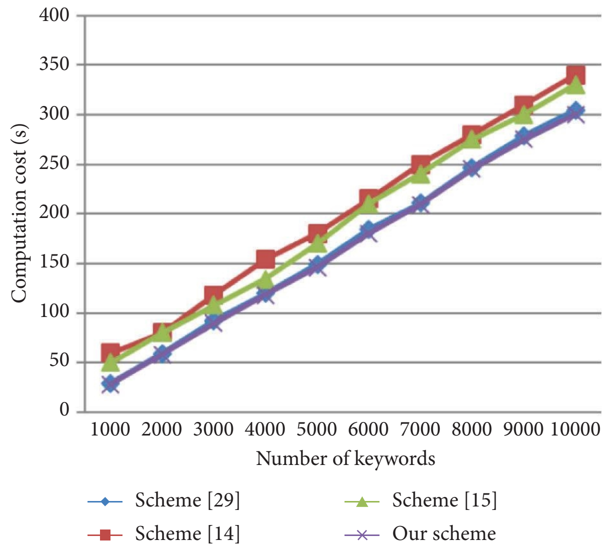
(a)



(b)

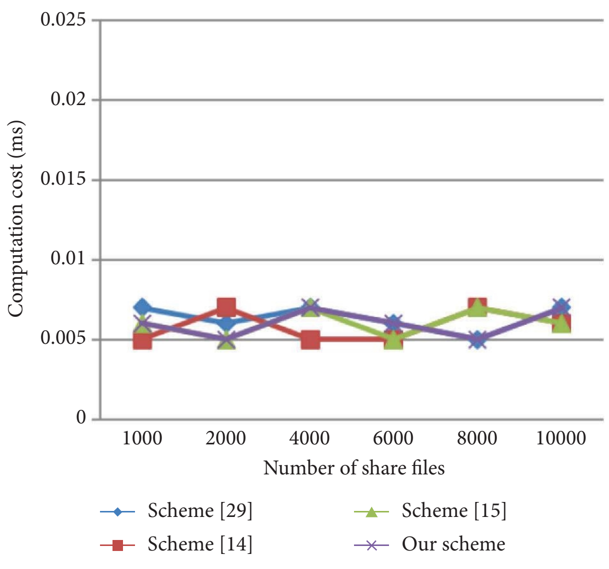


(c)

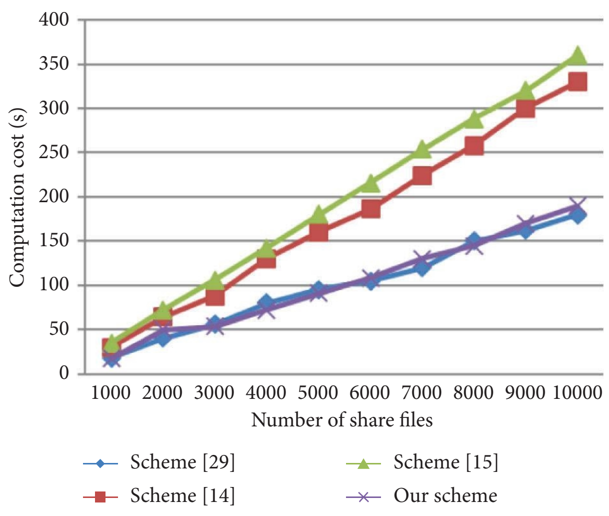


(d)

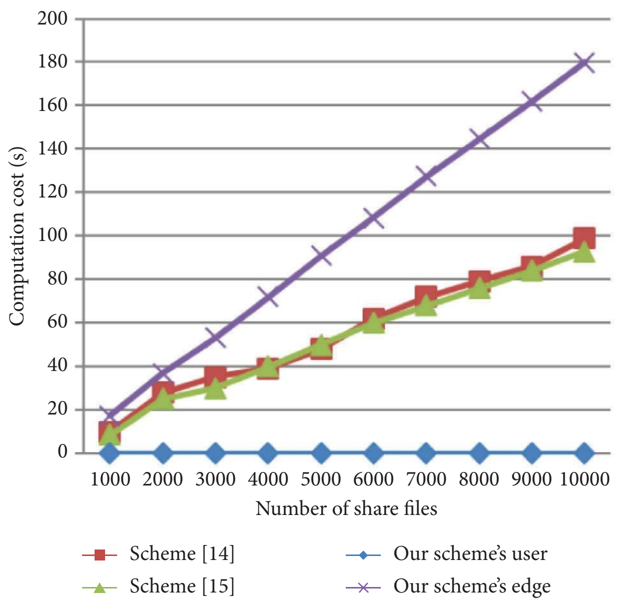
Figure 4: Continued.



(e)



(f)



(g)

Figure 4: Time cost comparison. (a) Time cost of Init. (b) Time cost of Keygen. (c) Time cost of Authorization. (d) Time cost of Encrypt. (e) Time cost of Trapdoor. (f) Time cost of search. (g) Time cost of verification.

In the Verification phase, Figure 4(g) shows that the user side of our scheme has less computational time than schemes [14, 15], which is because the edge completes most of the calculations, and the user only need to hash to get the verification results. Note that scheme [29] cannot achieve the verification function.

To sum up the experiments, the overall operation time of our scheme is lower than schemes [14, 15] with all processes.

8. Conclusion

In the edge computing environment, we proposed a new scheme of verifiable data search with fine-grained authorization, which not only realizes file-oriented search authority management but also verifies the correctness and completeness of results. In addition, with the assistance of edge server, clients with limited resource can easily carry out the tasks of search and verification. Besides, we proved that our scheme is verifiable and secure, and the secure model did not rely on the Random Oracle. To assess the practicability of the proposed scheme, we implemented it and conducted experiments in a simulated environment. As expected, our scheme effectively reduced the computation, communication, and storage costs. In the future, we intend to focus on the dynamic management of users and files, as well as the further authorization to prevent the key abuse. Also, we want to instantiate the data sharing model with the blockchain technology.

Data Availability

The processed data required to reproduce these findings cannot be shared at this time as the data also forms part of an ongoing study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research is supported by National Natural Science Foundation of China (No. 61932010) and Science and Technology Project of Guangzhou City (No. 201707010320).

References

[1] O. Osanaiye, K. K. R. Choo, and M. Dlodlo, "Distributed denial of service (ddos) resilience in cloud: review and conceptual cloud ddos mitigation framework," *Journal of Network and Computer Applications*, vol. 67, pp. 147–165, 2016.

[2] J. Shen, T. Zhou, X. Chen, J. Li, and W. Susilo, "Anonymous and traceable group data sharing in cloud computing," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 912–925, 2018.

[3] D. Xiaoding Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceeding of the 2000 IEEE Symposium on Security and Privacy*, IEEE, Berkeley, CA, USA, May 2000.

[4] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proceeding of the International conference on the theory and applications of cryptographic techniques*, Springer, Berlin, Heidelberg, June 2004.

[5] B. Wang, W. Song, W. Lou, and Y. Hou, "Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee," in *Proceeding of the 2015 IEEE Conference on Computer Communications (INFOCOM*, Hong Kong, China, May 2015.

[6] Y. Wang, S. Sun, J. Wang, J. K. Liu, and X. Chen, "Achieving searchable encryption scheme with search pattern hidden," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 1012–1025, 2022.

[7] Y. Yang, X. Liu, and R. H. Deng, "Multi-user multi-keyword rank search over encrypted data in arbitrary language," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 2, pp. 320–334, 2020.

[8] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[9] Y. Mansouri and M. A. Babar, "A review of edge computing: features and resource virtualization," *Journal of Parallel and Distributed Computing*, vol. 150, pp. 155–183, 2021.

[10] X. Wang, Y. Mu, R. Chen, and X. Zhang, "Secure channel free id-based searchable encryption for peer-to-peer group," *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 1012–1027, 2016.

[11] J. Shen, T. Zhou, X. Chen, J. Li, and W. Susilo, "Anonymous and traceable group data sharing in cloud computing," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 912–925, 2018.

[12] X. Liu, G. Yang, Y. Mu, and R. H. Deng, "Multi-user verifiable searchable symmetric encryption for cloud storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1322–1332, 2020.

[13] Z. Liu and Y. Liu, "Verifiable and authenticated searchable encryption scheme with aggregate key in cloud storage," in *Prceedings of the 2018 14th International Conference on Computational Intelligence and Security (CIS)*, Hangzhou, China, November 2018.

[14] X. Wang, X. Cheng, and Yu Xie, "Efficient verifiable key-aggregate keyword searchable encryption for data sharing in outsourcing storage," *IEEE Access*, vol. 8, pp. 11732–11742, 2020.

[15] Z. Liu, T. Li, P. Li, C. Jia, and J. Li, "Verifiable searchable encryption with aggregate keys for data sharing system," *Future Generation Computer Systems*, vol. 78, pp. 778–788, 2018.

[16] J. Shao, Z. Cao, X. Liang, and H. Lin, "Proxy re-encryption with keyword search," *Information Sciences*, vol. 180, no. 13, pp. 2576–2587, 2010.

[17] C. Wang, W. Li, Y. Li, and X. Xu, "A ciphertext-policy attribute-based encryption scheme supporting keyword search function." in *Cyberspace Safety and Security*, vol. 8300, pp. 377–386, Springer, 2013.

[18] P. Yanguo, C. Jiangtao, P. Changgen, and Y. Zuobin, "Certificateless public key encryption with keyword search," *China Communications*, vol. 11, no. 11, pp. 100–113, 2014.

[19] R. Zhang and R. Xue, "Efficient keyword search for public-key setting," in *Proceedings of the MILCOM 2015-2015 IEEE Military Communications Conference*, IEEE, Tampa, FL, October 2015.

[20] Y. Guo, F. Liu, Z. Cai, N. Xiao, and Z. Zhao, "Edge-based efficient search over encrypted data mobile cloud storage," *Sensors*, vol. 18, no. 4, p. 1189, 2018.

[21] Qi Chen, K. Fan, K. Zhang, H. Wang, H. Li, and Y. Yang, "Privacy-preserving searchable encryption in the intelligent edge computing," *Computer Communications*, vol. 164, pp. 31–41, 2020.

[22] Yu-C. Chen, X. Xie, P. S. Wang, and R. Tso, "Witness-based searchable encryption with optimal overhead for cloud-edge computing," *Future Generation Computer Systems*, vol. 100, pp. 715–723, 2019.

[23] S. Wang, C. Ding, N. Zhang et al., "A cloud-guided feature extraction approach for image retrieval in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 292–305, 2021.

[24] S. Ahmad, S. Zobaed, N. Raju, and M. Salehi, "Edge computing for user-centric secure search on cloud-based encrypted big data," in *Proceedings of the 2019 IEEE 21st international conference on high performance computing and communications; IEEE 17th international conference on smart city; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Zhangjiajie, China, August 2019.

[25] J. Li, J. Li, X. Chen, C. Jia, and Z. Liu, "Efficient keyword search over encrypted data with fine-grained access control in hybrid cloud," *NSS*, 2012.

[26] X. Liu, Y. Zhang, B. Wang, and J. Yan, "Mona: secure multi-owner data sharing for dynamic groups in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1182–1191, 2013.

[27] Z. Deng, K. Li, K. Li, and J. Zhou, "A multi-user searchable encryption scheme with keyword authorization in a cloud

storage," *Future Generation Computer Systems*, vol. 72, pp. 208–218, 2017.

[28] C. . Van Rompay, R. Molva, and Melek Önen, "Multi-user searchable encryption in the cloud," *Information Security. ISC 2015. Lecture Notes in Computer Science*, vol. 9290, 2015.

[29] B. Cui, Z. Liu, and L. Wang, "Key-aggregate searchable encryption (kase) for group data sharing via cloud storage," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2374–2385, 2016.

[30] R. Zhou, X. Zhang, X. Du, X. Wang, G. Yang, and M. Guizani, "File-centric multi-key aggregate keyword searchable encryption for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3648–3658, 2018.

[31] C. Jia, Z. Liu, T. Li, Z. Fu, and J. Li, "Key-aggregate searchable encryption under multi-owner setting for group data sharing in the cloud," *International Journal of Web and Grid Services*, vol. 14, no. 1, pp. 21–43, 2018.

[32] W. Zhang, Y. Lin, and Gu Qi, "Catch you if you misbehave: ranked keyword search results verification in cloud computing," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 74–86, 2018.

[33] X. Jianfeng Wang, C. Shifeng, S. Joseph, K. Liu, and M. Au, Z. Zhan, Towards efficient verifiable conjunctive keyword search for large encrypted database," *Computer Security. ESORICS 2018. Lecture Notes in Computer Science*, vol. 11099, 2018.

[34] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: keyed-hashing for message authentication," *Network Working Group*, vol. 2014, pp. 1–11, 1997.

[35] D. Boneh, X. Boyen, and Eu-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Proceedings of the annual international conference on the theory and applications of cryptographic techniques*, Springer, Berlin, Heidelberg, June 2005.

[36] M. Abdalla, M. Bellare, D. Catalano et al., "Searchable encryption revisited: consistency properties, relation to anonymous ibe and extensions," in *Proceedings of the Annual International Cryptology Conference*, Springer, Berlin, Heidelberg, May 2005.

[37] M. T. Goodrich, O. Ohrimenko, N. C. Triandopoulos, D. Nguyen, R. Tamassia, and C. V. Lopes, "Efficient verification of web-content searching through authenticated web crawlers," *Proceedings of the VLDB Endowment*, vol. 5, no. 10, pp. 920–931, 2012.