



Article

Blockchain-Assisted Verifiable and Multi-User Fuzzy Search Encryption Scheme

Xixi Yan, Pengyu Cheng ^{*}, Yongli Tang  and Jing Zhang

School of Software, Henan Polytechnic University, Jiaozuo 454000, China; yanxx@hpu.edu.cn (X.Y.); yltang@hpu.edu.cn (Y.T.); zhangj202227@163.cn (J.Z.)

* Correspondence: 212209020049@home.hpu.edu.cn

Abstract: Searchable encryption (SE) allows users to efficiently retrieve data from encrypted cloud data, but most of the existing SE solutions only support precise keyword search. Fuzzy searchable encryption agrees with practical situations well in the cloud environment, as search keywords that are misspelled to some extent can still generate search trapdoors that are as effective as correct keywords. In scenarios where multiple users can search for ciphertext, most fuzzy searchable encryption schemes ignore the security issues associated with malicious cloud services and are inflexible in multi-user scenarios. For example, in medical application scenarios where malicious cloud servers may exist, diverse types of files need to correspond to doctors in the corresponding departments, and there is a lack of fine-grained access control for sharing decryption keys for different types of files. In the application of medical cloud storage, malicious cloud servers may return incorrect ciphertext files. Since diverse types of files need to be guaranteed to be accessible by doctors in the corresponding departments, sharing decryption keys with the corresponding doctors for different types of files is an issue. To solve these problems, a verifiable fuzzy searchable encryption with blockchain-assisted multi-user scenarios is proposed. Locality-sensitive hashing and bloom filters are used to realize multi-keyword fuzzy search, and the bigram segmentation algorithm is optimized for keyword conversion to improve search accuracy. To realize fine-grained access control in multi-user scenarios, ciphertext-policy attribute-based encryption (CP-ABE) is used to distribute the shared keys. In response to the possibility of malicious servers tampering with or falsifying users' search results, the scheme leverages the blockchain's technical features of decentralization, non-tamperability, and traceability, and uses smart contracts as a trusted third party to carry out the search work, which not only prevents keyword-guessing attacks within the cloud server, but also solves the verification work of search results. The security analysis leads to the conclusion that the scheme is secure under the adaptively chosen-keyword attack.

Keywords: multi-user fuzzy search; smart contract; access control



Citation: Yan, X.; Cheng, P.; Tang, Y.; Zhang, J. Blockchain-Assisted Verifiable and Multi-User Fuzzy Search Encryption Scheme. *Appl. Sci.* **2024**, *14*, 11740. <https://doi.org/10.3390/app142411740>

Academic Editor: Gianluca Lax

Received: 3 November 2024

Revised: 7 December 2024

Accepted: 13 December 2024

Published: 16 December 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Due to the convenience of cloud services, individuals and organizations are inclined to upload their files to the cloud for storage, it is crucial to encrypt sensitive files before storing on the cloud. However, another problem arises as to how to perform search operations on the ciphertext files in cloud services. Searchable encryption technology [1] perfectly solves the problem of searching ciphertext files directly and has a wide range of applications, such as in the smart healthcare [2], smart city, and IoT fields [3].

Since the pioneering work on searchable encryption, scholars have devoted themselves to studying some practical and feasible searchable encryption schemes. For example, a scheme [4] supports conjunctive, subset and range queries on encrypted data, a scheme [5] supports spatial keyword queries, and another scheme [6] supports Boolean queries. In practice, when users perform a search operation, they often misspell the search keywords or simply enter the approximate stem of the keyword or other forms to start performing

the search operation. To address this situation, fuzzy search [7] realizes a certain degree of approximate keyword search matching, which can generate the same or similar keyword trapdoor for users to perform search matching. However, a few existing fuzzy search schemes still have limitations. For example, the schemes in [7] and the scheme in [8] only support single keyword search. Although there exists a multi-user scenario implemented in the scheme in [9] and the scheme in [10], where multiple users can retrieve the ciphertext through a single cloud server. In the case that multiple users have search permissions, it is necessary to ensure that people with different search permissions can search for the corresponding ciphertext files. Therefore, fine-grained access control needs to be ensured. There is the problem that the encryption and decryption keys are difficult to share, the online auditing and authorization by administrators for users applying for access are not friendly when it comes to the actual situation of smart healthcare, and do not satisfy the fine-grained access control of multi-user application scenarios. Although the scheme [11] can search for ciphertext files with corresponding permissions in multi-user scenarios, current fuzzy search schemes seldom consider fine-grained access control on shared keys for different types of files. Therefore, it is essential to provide multi-keyword fuzzy search encryption schemes with fine-grained control under multi-user conditions.

In the scenario of smart healthcare, doctors in different departments need to search for the ciphertext of case files. The case information of different patients will be first classified in a certain way, and then the medical data center will store the patient's case and other private information through encryption in the medical database or cloud server, which can only be accessed by authorized doctors. In practice, doctors in different departments have access rights to different files, e.g., doctors in the cardiology department can only access cases of cardiac patients. Although some fuzzy search schemes support multiple user scenarios, there is the problem of difficulty in sharing encryption and decryption keys, and it is unfriendly in the actual situation in smart healthcare to review and authorize the users applying for access online through administrators.

In addition, some scheme [12–15] models serve as honest but curious entities, which are not realistic in practice. Cloud servers have too much power and are not conducive to the data owner's control over the data. Some malicious cloud servers even seriously harm the data owner's interests, such as returning wrong or incomplete files. This greatly affects the practical application of fuzzy search encryption schemes. Therefore, a decentralized fuzzy search encryption scheme is necessary and feasible to solve the possible problems of malicious cloud servers.

Aiming at the above problems, the Blockchain-Assisted Verifiable and Multi-User Fuzzy Search Encryption Scheme is proposed, which has more comprehensive functions and efficient performance. And it is more adaptable to the ciphertext search needs under new network services such as smart healthcare. The main innovations are as follows.

- For the existence of malicious cloud servers in fuzzy search schemes, semi-decentralization is used to weaken the power of cloud servers, a smart contract is constructed to guarantee fairness between users and servers, and blockchain is introduced to assist in achieving integrity verification of results.
- Aiming at the lack of fine-grained access control for users in one-to-many application scenarios, the is used to distribute the shared key with a tree structure, and only authorized users who meet the access results can decrypt the shared key.
- By improving the index tree structure and bigram participle algorithm, the proposed scheme not only optimizes the search efficiency and search precision but also enables users to check the search results locally for a second time.
- Through security analysis, it is proved that the scheme is secure under the adaptive selection of keyword attacks. Functional comparison and experimental analysis show that the search efficiency of the scheme has been improved under the premise of supporting multi-functionalities such as multiple keywords, fuzzy search, dynamic updating, and result validation. In particular, the search time of our scheme is more advantageous with the increase in search keywords and searched files.

The rest of this paper is arranged as follows. Related work on fuzzy search and blockchain-based searchable encryption is in Section 2. Some preliminaries used in the scheme are given in Section 3. In Section 4, the definitions of algorithm and security model are provided. Details of the program will be presented in Section 5. Sections 6 and 7 provide the program analysis and efficiency analysis, respectively. Section 8 summarizes the work and gives conclusions.

2. Related Work

We present research work on fuzzy searchable encryption and blockchain-based searchable encryption in this section.

2.1. Fuzzy Searchable Encryption

Since the fuzzy search encryption scheme has a broader application prospect, it attracts scholars to study it continuously. Li et al. [7] first formalized the fuzzy keyword search problem on encrypted cloud data and designed a wildcard-based secure index construction scheme, which first constructs a wildcard-based fuzzy keyword set, each element in the set is encrypted, and when searching, the user first generates the same encrypted fuzzy keyword set, then the cloud server finds the corresponding keywords, and finally returns the corresponding files. Subsequently, Wang et al. [16] improved on the scheme in [7] and proposed a searchable encryption scheme for fuzzy keyword search based on the index tree structure of keywords, which improves the search efficiency, but it only supports single keyword search. Although [17] designed a dual ranking function that combines keyword weights and keyword morphological similarity to rank search results, it still cannot avoid the limitations of single-keyword search.

Kuzu et al. [18] scaled by minhash to transform keyword vectors and determine the similarity of keywords. Subsequently, Wang et al. [19] implemented a multi-keyword search based on scheme [18]. Fu et al. [14] improved the search accuracy with the Bloom filter and location-sensitive hash function but still did not support the dynamic update function. Zhong et al. [15] proposed a scheme to support efficient dynamic updating, which improves the search accuracy and supports encrypted index tree based on the top-k sorting of encrypted files. Tong et al. [20] designed a dual bloom filter to store and mask the keywords contained in a file and realized the verification of correctness and completeness, but the verification is more complicated. Li et al. [21] designed an improved bi-gram participle algorithm to enhance fuzzy search accuracy, the scheme assumes cloud server is malicious but fails to construct an effective solution for the malicious behavior of malicious servers.

2.2. Blockchain-Based Searchable Encryption

The cloud server has too much right in traditional searchable encryption schemes, and all the search processes are dependent on the cloud server. Scholars have introduced the blockchain fairness mechanism to realize fairness due to the non-tamperable characteristics, which guarantees that if a user and cloud server follow the agreement honestly, the cloud server gets the corresponding service fee, while users get the correct results. Once a cloud server is detected with dishonest behavior, it will not get the service fee and will be punished with the loss of margin.

Cai et al. [22] realized dynamic keyword search in distributed storage combined with blockchain technology for fair search. Wang et al. [23] constructed a decentralized privacy-preserving search scheme. Data owners can be assured of receiving correct results without having to worry about malicious server behavior. Hu et al. [24] combined blockchain technology to achieve fair payment and constructed a file-sharing scheme in one-to-one as well as one-to-many user scenarios. Smart contracts are used to store secure indexes and execute searches, ensuring that users get correct results if they pay for the transaction. Chen et al. [25] constructed a blockchain-enabled public key encryption scheme with multi-keyword search, which uses smart contract to ensure the fairness without introducing

a third party. Guo et al. [26] constructed an authentication-enabled search encryption scheme on blockchain and implement forward security. It is evident that the combination of blockchain and searchable encryption cryptography to achieve decentralization, build fair transactions, and authentication features is effective.

3. Preliminaries

We introduce the technical knowledge used in fuzzy search in this section.

3.1. Bloom Filter (BF)

The Bloom Filter is a space-saving probabilistic data structure that determines whether an element is in a collection or not. Mapping two or more different elements to the same position in a bit array via a hash function has a false positive rate.

Suppose the bloom filter initializes a p -bit array, k random hash functions, and the element set $\partial = \{\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n\}$.

For each element in the set, it is mapped to the k positions of the p – bit bloom filter by k independent hash functions and corresponding positions are set to 1. When determining whether α_x is in ∂ , it uses the same approach as described above to map α_i to the k positions of the bloom filter. If one of the k positions has a value of 0, then $\alpha_x \notin \partial$; if all k positions have a value of 1, then $\alpha_x \in \partial$ or there is a false positive.

3.2. Locality Sensitive Hashing (LSH)

In the scheme, the LSH determines whether the keyword is in the file or not by mapping keywords into a hash bucket. A positional hash function is defined as follows: A family of hash functions F is a mapping from Euclidean space S_2 to hash coding space U . The hash function F is said to satisfy (r_1, r_2, p_1, p_2) – ness when $Pr_F[h(q) = h(p)] \geq p_1, p \in B(q, r_1)$ and $Pr_F[h(q) = h(p)] \leq p_2, p \in B(q, r_2)$ are satisfied, where B is the space centered at q with radii r_1, r_2 . The p -stable position-sensitive hash function is calculated as $h_{a,b}(v) = \lfloor \frac{a \cdot v + b}{n} \rfloor$, where the parameters a and b are a d -dimensional vector and a random number between $[0, n]$, respectively, and each element in the vector v satisfies p -stable distribution. As shown in Figure 1, $a \cdot v$ means that vector a is mapped onto an axis with vector v as the base vector, and this axis is divided into n equal parts of m . The labelled value of the interval corresponding to the value of $a \cdot v$ is taken as the hash value of vector a . We can determine whether a keyword can be mapped to a bucket by calculating the probability that the values are equal by using LSH.

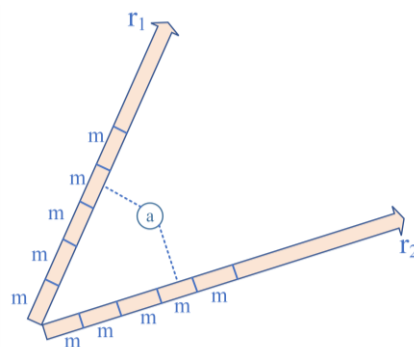


Figure 1. p -stable element mapping.

4. Definitions of Algorithm and Security Model

Before describing our scheme in detail, the basic model, algorithmic framework, and security model are first given.

4.1. System Model

The model is based on the following roles, as shown in Figure 2.

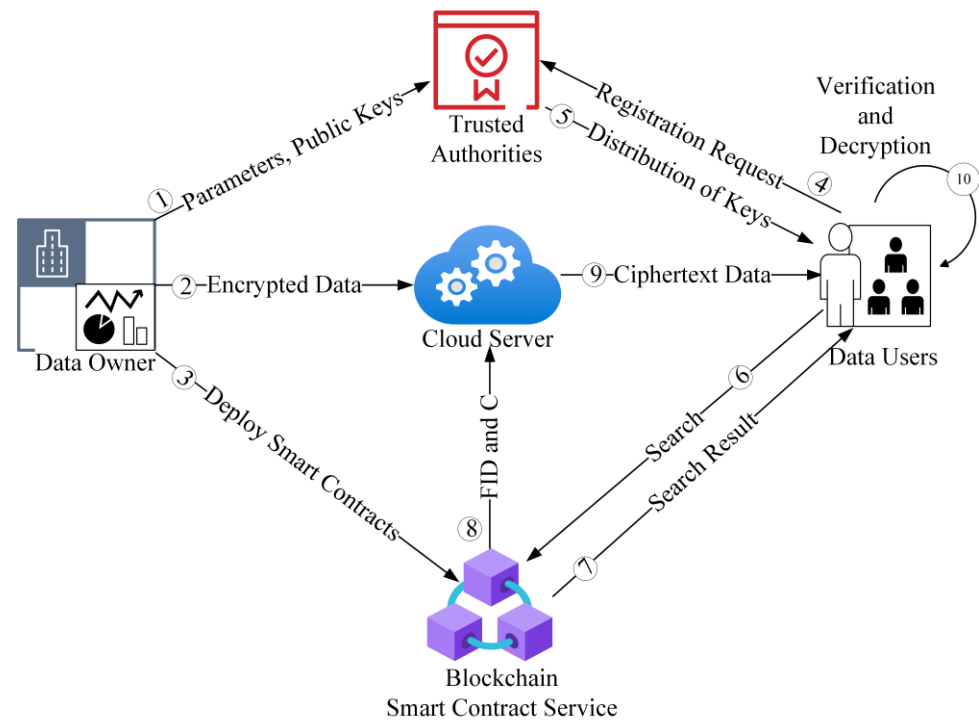


Figure 2. System model.

- (1) **Data Owner (DO).** *DO* extends beyond individual entities to data centers that are responsible for the integration of file information owned by multiple data owners. This implementation realizes the application of a multi-owner scenario from another perspective.
- (2) **Trusted Authority (TA).** As an authenticating entity, it generates attribute private keys for data users and transmits system public parameters and other relevant information to the users securely.
- (3) **Data Users (DU).** *DU* need to apply for their attribute private keys based on their attributes as well as reliable information, then generate and send search trapdoor to the smart contract account for searching transactions through query requirements.
- (4) **Smart Contract (SC).** As a decentralized component of the system, *SC* stores secure index tree $Index_{tree}$ and verification information of data owners, mitigating the excessive and uncontrolled authority of the cloud server. The search contract performs the initial segment of the entire search transaction and ensures the verification process for *DU*.
- (5) **Cloud Server (CS).** *CS* has powerful storage as well as computational capabilities, and stores the ciphertext data files outsourced by *DO*, and provides *DU* with the second part of the search service based on the transaction initiated, it may have the malicious behavior of tampering with the user's search results.

The workflow of the system is as follows.

DO transmits parameters, public keys, etc., to *TA* initially (Step 1), firstly categorizes all large collections of files, encrypts classified file collection $D = \{F_1, F_2, \dots, F_n\}$ and outsources the ciphertext set $C = \{C_1, C_2, \dots, C_n\}$ to *CS* (Step 2). To achieve access control and efficient search, *DO* constructs a file index tree and sends the index tree to the contract storage address (Step 3). *DU* send a registration request to *TA* (Step 4) and receives attribute private keys and search keys (Step 5). Then the *DU* generates a search token locally based on query keywords and algorithms disclosed by *TA* and sends it through a transaction to *SC* (Step 6). *SC* executes a search to obtain a search list and sends the information to the user (Step 7). The ciphertext of file identifiers (*FID*), the ciphertext set $C = \{C_1, C_2, \dots, C_n\}$ and transaction information is sent to *DU* (Step 8). Then *CS* returns to *DU* the ciphertext

data of the search transaction (step 9). *DU* can verify the search results locally, and only users whose attribute private key satisfies the access control can finally decrypt the file and malicious servers will receive penalties. (Step 10).

4.2. Algorithm Framework

4.2.1. Definition

Before giving the definition of the algorithm, the definitions of the symbols used in the scheme are given in Table 1.

Table 1. Description of Symbols.

Notation	Meaning
$D = \{F_1, F_2, \dots, F_n\}$	the file set of <i>DO</i>
$C = \{C_1, C_2, \dots, C_n\}$	the ciphertext set
pk	the public key
Γ	the access policy
K_i	the symmetric key for the corresponding type of files.
C_{K_i}	the key cipher containing access policy
K_2	the verification key
S_u	the user attributes collection
W_q	a query keyword collection
KS	the index encryption key
SK_u	attribute private key for users
SK_s	the user search key
T_t	a trapdoor with time stamp
KW	a collection of keywords for a file set
KW_{F_i}	the keyword set for file F_i
ST_{kw}	the stemming of the keyword kw
$ST_{KW_{F_i}}$	the keyword stemming set for file F_i
MAC_{C_i}	the ciphertext message authentication code
$KS = (M_1, M_2, S)$	the key set for the encrypted trapdoor and index

4.2.2. Algorithm Definition

The scheme has nine main algorithms.

- $Setup(1^\lambda) \rightarrow (pk, mk)$ is a system initialization algorithm. The Data Owner *DO* inputs random security parameter λ , generates system public key pk and the master private key mk .
- $KeyGen(1^\lambda, l) \rightarrow KS$ is a key generation algorithm. *DO* inputs random security parameter λ and the length l of file index, outputs the key set $KS = (M_1, M_2, S)$.
- $Enc(pk, K_1, K_2, D, KW, KS, \Gamma) \rightarrow (FID, C, C_{K_1}, I, MAC_{C_i}, Index_{tree})$ is an encryption algorithm. *DO* takes public key pk , plaintext set D , keyword set KW , access policy Γ , key K_i of the corresponding type of files, authentication key K_2 , and index encryption key KS as input. Then, for outputs, the ciphertext set C , encrypted file identifier FID , security index I , the key cipher containing access policy C_{K_i} , ciphertext file message authentication code MAC_{C_i} , and security index tree $Index_{tree}$.
- $UserSign(pk, mk, S_u) \rightarrow (SK_u, SK_s, user_id)$ is a user registration algorithm run by *TA*, it takes pk, mk , and S_u as input, outputs attribute private key SK_u , the user search key SK_s , and authenticated identifier $user_id$.
- $TokenGen(pk, SK_s, W_q) \rightarrow T_t$ is a trapdoor generation algorithm. Data User takes the public key pk , a query keyword collection W_q and attribute private key SK_s as input, outputs the trapdoor with time stamp T_t .
- $Search(Index_{tree}, T_t) \rightarrow (C_{K_i}, FID, MAC_{C_i})$ is a search-matching algorithm run by Smart Contract *SC*. This algorithm takes T_t and $Index_{tree}$ as input, outputs C_{K_i}, MAC_{C_i} and FID .
- $Verify(C_i, MAC_{C_i}, K_2) \rightarrow 1/0$ is a verification algorithm. The Data User takes the verification key K_2 , the ciphertext file C_i and the ciphertext message authentication

code MAC_{C_i} as input. It is determined whether a preliminarily correct result is obtained by outputting the result.

- (8) $Dec(SK_u, C_{K_i}, C_{w_q}) \rightarrow (D_{kw} / \perp)$ is a decryption algorithm run by the Data User. It takes the attribute private key SK_u , the key cipher containing access policy C_{K_i} , and file set cipher C_{w_q} , which are based on the query keyword set w_q as input. If SK_u satisfies the access structure Γ defined by the Data Owner DO , then it outputs the plaintext files containing the query keyword set, otherwise, the output is \perp .
- (9) $Update(Uptype, I_{T_c}) \rightarrow Index_{tree_new}$ is an index tree update algorithm run by the smart contract. It takes the encrypted subtree I_{T_c} sent by the Data Owner as input, and outputs the new index tree $Index_{tree_new}$. The detailed process is described in Section 5.2.4.

4.3. Security Model

The CS is considered an entity that may have malicious behavior and is not fully trustworthy, it may return untrue results to save computation and bandwidth resources, which can seriously harm the DU . In addition, DU uploads its own set of attributes honestly to obtain $user_id$, all communication with TA is secure.

Regarding privacy requirements, this article considers two threat models with different adversary capabilities.

Known Ciphertext Model: The cloud server can only observe the encrypted file set, the encrypted index, the submitted tokens, and the search results. It has no prior knowledge of the file set or the search keywords.

Known Background Model: the cloud server is also able to obtain some additional background information like statistical information about the files, the queried keywords and their corresponding tokens, and the frequency of searched keywords.

A simulation-based game between adversary \mathcal{A} and a stateful simulator \mathcal{B} using an allowed leakage access model and a search model are used to prove security. Two leakage functions $\mathcal{L} = (\mathcal{L}_1, \mathcal{L}_2)$ are defined, where \mathcal{L}_1 and \mathcal{L}_2 are expressed formally as follows.

$\mathcal{L}_1(D) = \{|D|, n, \{|F_i|, ind(F_i)\}_{i \in [1, n]}\}$ and takes the plaintext file collection D as input and outputs the plaintext collection size $|D|$, the number of files n , the size of each file $|F_i|$, and file identifiers $ind(F_i)_{i \in [1, n]}$.

$\mathcal{L}_2(D, W_q^*) = \{P(W_q^*), T^*\}$, it takes the plaintext file collection D and the search keyword collection as input and outputs the keyword access patterns $P(W_q^*)$ and search trapdoor T^* . Here, suppose \mathcal{A} is an adversary, \mathcal{B} is a stateful simulator, and \mathcal{G} acts as a challenger. The interaction game between \mathcal{A} , \mathcal{B} , and \mathcal{G} is as follows.

- $Real_{\mathcal{A}}(\lambda)$. \mathcal{G} initializes system and runs $KeyGen(1^\lambda, m) \rightarrow KS$, then sends a collection of files D to \mathcal{G} . \mathcal{G} generates encrypted file C and index I with a secure symmetric encryption algorithm and index generation algorithm, then sends them to \mathcal{A} . \mathcal{A} generates a polynomial number of keywords sets and launches an adaptive search $Q_1 = (w_1, w_2, \dots, w_q)$. Then the search keywords in set are transformed into a set of vectors to generate search trapdoor by \mathcal{G} , which is sent to \mathcal{A} . Eventually, \mathcal{A} outputs $answer \in \{0, 1\}$, ending the game.
- $Ideal_{\mathcal{A}}(\lambda)$. Given the leakage functions \mathcal{L}_1 and \mathcal{L}_2 , \mathcal{G} inputs plaintext file D . Simulator \mathcal{B} generates a simulated index I^* and ciphertext C^* , sends I^* and C^* to \mathcal{A} . \mathcal{B} chooses a polynomial number of keywords $Q_1 = (w_1, w_2, \dots, w_q)$ for adaptive interrogation, then outputs search trapdoor T_i^* based on \mathcal{L}_2 . Finally, \mathcal{A} outputs the result $answer' \in \{0, 1\}$, ending the game.

5. Proposed Scheme

We describe the main ideas of fuzzy search and detail the algorithms in this section.

5.1. Main Idea

The scheme implements fuzzy search based on LSH and BF. Assuming that DO has n plaintext files $D = \{F_1, F_2, \dots, F_n\}$, a p bit bloom filter B_i is generated as an index vector, which contains all the keywords in F_i . To map the keywords in F_i on B_i , the keywords need to be converted into bigram vectors, then the bigram vectors are mapped to B_i by LSH. Trapdoor generation similarly requires transforming the query keyword W_q into a bigram vector and generating an m – bit bloom filter B_{w_q} , which maps bigram vector on B_{w_q} using the same LSH to generate index vector. The inner product of B_i and B_{w_q} represents the relevance of file F_i and query w_q . As shown in Figure 3.

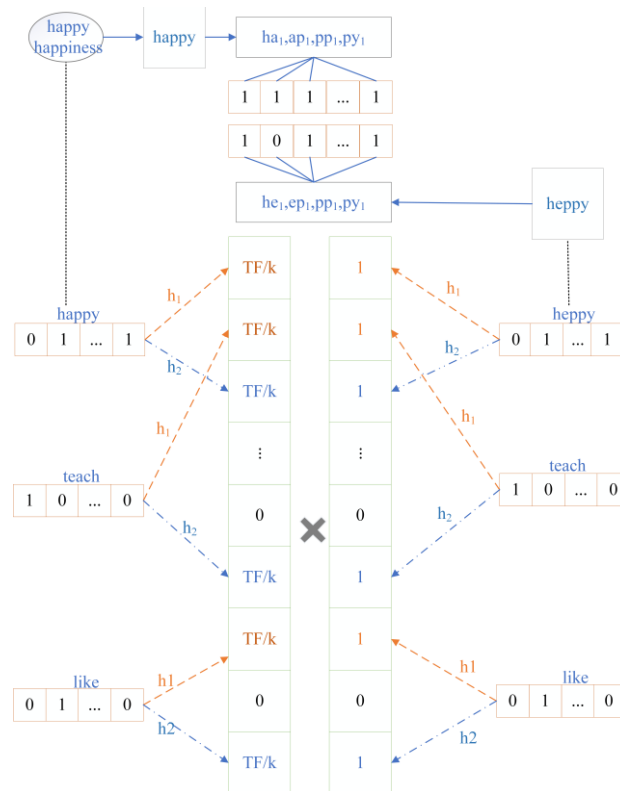


Figure 3. LSH-Bloom Architectural Diagram.

5.2. Concrete Construction

We describe the various algorithms and talk about them in four broad modules, namely, the preliminary work, the encryption work, the search work, and the subsequent work.

5.2.1. Preliminary Work

The preliminary work includes $Setup(1^\lambda)$, $KeyGen(1^\lambda, m)$ and $UserSign(pk, mk, S_u)$.

$Setup(1^\lambda)$: DO defines two multiplicative cyclic groups G_0, G_1 on \mathbb{Z}_p , the bilinear map $e : G_0 \times G_0 \rightarrow G_1$, randomly selects anti-collision hash function $H_1 : \{0, 1\}^* \rightarrow G_0$, and two pseudo-randomized functions $H_2 : \{0, 1\}^l \times \{0, 1\}^* \rightarrow \{0, 1\}^k$ and $H_3 : \{0, 1\}^l \times \{0, 1\}^* \rightarrow \{0, 1\}^k$. g is the generating element of G_0 , p is a large secure prime, and $e(g, g)$ is the value of the bilinear mapping in the swarm. TA selects α, β randomly as input, outputs the master private key $mk = (\alpha, \beta)$, and $pk = (G_0, G_1, p, g, h = g^\beta, e(g, g)^\alpha, H_1, H_2, H_3)$.

$KeyGen(1^\lambda, l)$: DO inputs a random security parameter λ and the index length l to generate $KS = (M_1, M_2, S)$ to encrypt the trapdoor and index, where $(M_1, M_2) \in R^{l \times l}$, $S \in \{0, 1\}^l$ is a vector that prevents brute force decryption and M_1, M_2 are invertible matrices.

$UserSign(pk, mk, S_u)$: When a user sends an enrollment request, TA checks the user's attribute set S_u , then confer $user_id$ and search key SK_s to the user.

5.2.2. Encryption Work

The encryption work mainly includes key encryption, file index vector generation, index tree generation, and trapdoor generation.

A. Key Encryption

$Enc(pk, K_1, K_2, D, KW, KS, \Gamma)$: Assume that the file set has t types, DO has n plaintext files $D = \{F_1, F_2, \dots, F_n\}$. The files are categorized according to their types and the corresponding symmetric encryption key $K_i (1 \leq i \leq t)$ is generated for different types of files., e.g., in a medical database, ophthalmology case files are classified as one type and case files from different sections are categorized into different types. DO parses keyword sets KW from D and KW_{F_i} from F_i respectively, then encrypts file identifiers of $D = \{F_1, F_2, \dots, F_n\}$ by K_2 to obtain the ciphertext $FID = \{FID_1, FID, \dots, FID_n\}$.

DO sets the access control structure tree Γ for different types of files with key K_1 based on actual application requirements. For example, the core files of a medical department can be decrypted only by users who meet the conditions of important attributes, while for the general files of the department only the basic conditions of access to the files by the department need to be met. DO assign a polynomial q_x for each node x from root r , where q_x in leaf nodes are constant. Randomly select s from \mathbb{Z}_p which $q_r(0) = s$, $\deg(q_x) = d_x = k_x - 1$, then $deg(q_r)$ random coefficients are selected from \mathbb{Z}_p to determine q_x . Y denotes the leaf nodes and k_x is the threshold value of x . Eventually gets $C_{K_i} = \{\Gamma, \bar{C} = K_1 e(g, g)^{as}, C = h^s, \{C_y, C_{y'}\} \forall y \in Y\}$ where $C_y = g^{q_y(0)}$, $C_{y'} = H_1(att(y))^{q_y(0)}$. DO encrypts plaintext $D = \{F_1, F_2, \dots, F_n\}$ by K_i to obtain ciphertext set $C = (C_1, C_2 \dots, C_n)$. Eventually, C and FID are sent to CS , C_{K_i} will be stored in the index tree described subsequently.

B. File Index Vector Generation

The generation of file index vectors is carried out by mapping the keywords on the bloom filter of the file. The keywords transform into vectors so that they can be mapped on the bloom filter and the improved keyword bigram disambiguation algorithm from scheme [21] is used to transform the keywords into vectors.

For the keyword "happy", DO first converts it to a bigram set $\{ha_1, ap_1, pp_1, py_1\}$. Considering the existence of the same bigram elements, the vector is expanded to 2×26^2 , where 26×26 represents possible binary letters. If the bigram set of the keyword exists in the corresponding position of the vector, the corresponding position is set to 1, otherwise it is set to 0, different keywords are converted into vectors with the same length.

DO constructs and initializes an l – bit bloom filter B_i for file F_i in the file collection D by k LSH. I' and I'' are l – bit null vectors. The keyword set KW_{F_i} is subjected by the porter stemming algorithm to obtain the stemmed set $KW_{ST_{F_i}}$. If the file F_i has x keywords, the bigram set of $KW_{ST_{F_i}}$ finally converts to a vector set $Bv_{ST_{F_i}}$, bv_i in the vector set $Bv_{ST_{F_i}}$ are mapped onto B_i by k LSH. Set the corresponding position of the mapping to $TF_{i,j}/k$ according to the importance of the file according to the importance of the keyword to the file, where $TF_{i,j} = 1 + |kw_j|/|F_i|$ is the frequency of keyword kw_j in F_i , $|kw_j|$ denotes the number of occurrences of kw_j in F_i , and $|F_i|$ denotes the number of keywords in F_i .

The index vector construction for F_i is demonstrated by an example, as shown in the left side of Figure 3. Suppose F_i has three keywords "happily, teachers, likely", where $S = [1, 0, 1, 0, 0, 1, 1, 0, 0, 1]$, $l = 10$, M_1, M_2 are invertible matrices and $M_1, M_2 \in R^{10 \times 10}$. DO constructs an l – bit bloom filter B_i for F_i , then converts the set of keywords to the stem set $ST_{kw} = \{happy, teach, like\}$. The stem keyword "happy" is first converted to $Bv_1 = \{ha_1, ap_1, pp_1, py_1\}$ and then converted to vector bv_1 . For ease of representation, Figure 3 is represented by a string of 0 and 1 vectors. Then Bv_2, Bv_3 and bv_2, bv_3 are constructed for the rest stem keywords in the same way, finally, bv_1, bv_2 , and bv_3 are mapped as inputs to k LSHs on B_i , respectively, and the corresponding position of B_i is set to $TF_{i,j}/k$. Assuming three keywords have the same weight $TF_{i,j}/k$ of 0.5, the final index vector B_i of the file F_i is obtained.

C. Index Tree Generation

Denote S in KS and bloom filter B_i of file F_i as $S = (s_1, s_2, \dots, s_m)$ and $B_i = (b_1, b_2, \dots, b_m)$, respectively, where B_i and S have the same structure. For each b_i in B_i , let $B_i' = (b_1', b_2', \dots, b_m')$, $B_i'' = (b_1'', b_2'', \dots, b_m'')$. If $s_i = 1$, then $b_i = b_i' = b_i''$; If $s_i = 0$, then $b_i' = b_i/2 + r$, $b_i'' = b_i/2 - r$, where r is a random parameter and $r \in R$. Finally, the encrypted index $I_i = (I_i', I_i'')$ of the file F_i is obtained, where $I_i' = (M_1^T \cdot B_i')$, $I_i'' = (M_2^T \cdot B_i'')$.

The index tree construction of the scheme in [15] is used to construct the $Index_{tree}$ and the storage structure of the leaf nodes is improved on this basis. Leaf nodes store file identifiers and vectors in addition to filing authentication codes and key ciphertexts with access control structures. The internal nodes store the optimal vectors of their leaf node index vectors and the internal node vectors are encrypted in accordance with the index encryption method of F_i . Finally, the $Index_{tree}$ is sent to SC , the $Index_{tree}$ is shown in Figure 4.

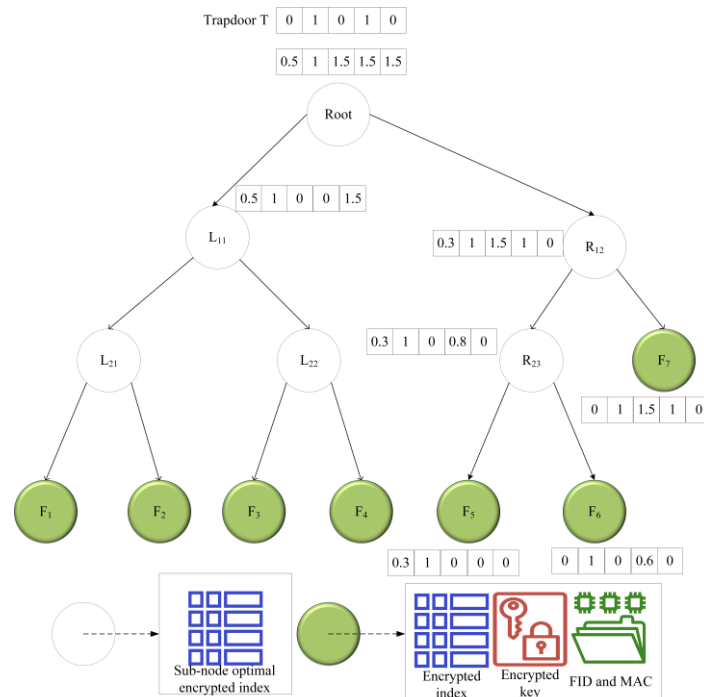


Figure 4. The index tree.

D. Trapdoor Generation

$TokenGen(pk, SK_s, W_q)$: DU first converts the set of keywords W_q into a bigram set by the bigram segmentation algorithm, then the bigram set is transformed into a binary vector set Bv using the same method as for index generation. Each vector in Bv is used as an input to the k LSHs, respectively, to map into an l – bit bloom filter, the value of the mapped location is set to 1. Finally, the vector B_t of the keywords is obtained.

The bloom filter used to generate the search trapdoor is denoted as $B_t = (t_1, t_2, \dots, t_m)$, S in KS is denoted as $S = \{s_1, s_2, \dots, s_m\}$. S and B_t have the same structure as $B_t' = (t_1', t_2', \dots, t_m')$ and $B_t'' = (t_1'', t_2'', \dots, t_m'')$. For each t_i in B_t , if $s_i = 1$, then $t_i' = t_i/2 + r$, $t_i'' = t_i/2 - r$; if $s_i = 0$, then $t_i = t_i' = t_i''$. Then, encrypt B_t', B_t'' to get $T' = (M_1^{-1} \cdot B_t')$, $T'' = (M_2^{-1} \cdot B_t'')$. Eventually, the secure search trapdoor $T = (T', T'')$, $T_t = (T || L_t)$, and $user_id$ are sent to SC through transaction.

5.2.3. Search Match

$Search(Index_{tree}, T_t)$: DO defines a reasonable search unit price $\$search$, then sets $\$pro_DU$ as the deposit of DU stored temporarily in SC to prevent DU from quitting in the middle. $\$All$ represents the total price that DU should pay for each search transaction and $\$search_price$ represents the overhead of calling the search algorithm. gas_lim and

gas_price respectively represent the maximum gas consumption and gas unit price in the SC, and the product of the two represents the maximum price that the SC is allowed to accept.

SC obtains the user's search trapdoor and $user_id$ through the transaction. After authenticating whether the identity is correct, it checks whether the pre-deposit of DU satisfies the search transaction, if it does, a search match will be performed, otherwise, the transaction is canceled.

The index tree will be traversed from top to bottom. Meanwhile, SC calculates $T^T \cdot I_i$, which represents the correlation score $Score$ of the node and search keywords, where $T \cdot I_i = \{M_1^{-1}B_{i'} \cdot M_1^T B_{i'} + M_2^{-1}B_{i''} \cdot M_2^T B_{i''}\}$. If the $Score$ of the parent node vector and the search trapdoor is less than the threshold TH , the traversal of the node's children is stopped. The entire index tree is traversed in this way.

SC defines a list $Sclist$, which consists of at most k_n records. Each entry contains a pair $\langle Score_{F_i}, value \rangle$, where $value = \{FID || C_{K_i} || MAC_{C_i}\}$, $Score = \{T * I_i\}$. When traversing to the leaf node, SC stores the information of leaf node in $Sclist$. After traversing the index tree, SC sorts and eliminates the k_n entries of $Sclist$ based on $Score_{F_i}$. Finally the k entries that match the user's search are obtained, then SC sends the $value$ in $RList$ to DU, sends the FID and transaction information in $value$ to the CS. After receiving the message, CS packages the ciphertext set C and sends it to the DU.

SC uses GDFS (Greedy Depth-First Search) for searching, the notation and Algorithm 1 used are described as follows.

$Score_{nd}$: correlation score between index vector I_i of node nd and search vector T .

$nd.hclld$: child nodes of node nd with high query relevance.

$nd.lclld$: child nodes of node nd with low query relevance.

Algorithm 1 Search Matching Algorithm

Input: the index tree $Index_{tree}$

Output: $Sclist$

```

1: if  $nd$  is not a leaf node then
2:   if  $Score_{nd} > TH$  then
3:     GDFS ( $nd.hclld$ )
4:     GDFS ( $nd.lclld$ )
5:   else
6:     return
7:   end if
8: else
9:   if  $Score_{F_i} > TH$  &&  $Sclist$  is full
10:    delete the lowest relevance node from  $Sclist$ 
11:    insert the new node  $\langle Score_{F_i}, value \rangle$  and sort all the node by  $Score_{F_i}$ 
12:   else if  $Score_{F_i} > TH$  &&  $Sclist$  is not full
13:    insert the new node  $\langle Score, value \rangle$  in  $Sclist$ 
14:   end if
15:   return
16: end if

```

As the child nodes are overwritten by parent nodes, only part of the tree needs to search. As in Figure 4, the file collection $D = \{F_1, F_2, \dots, F_7\}$, the threshold TH is set to 1.5, and the trapdoor $T = (0, 1, 0, 1, 0)$ is sent to SC by the user. Assume that 1 most relevant file for the search is selected.

SC traverses the nodes from top to bottom and computes $Score$ for each node. The children of L_{11} do not need to traverse due to $Score_{Root} = 2.5 > TH$, $Score_{L_{11}} = 1 < TH$, $Score_{R_{12}} = 2 > TH$, $Score_{R_{23}} = 1.8 > TH$, $Score_{F_5} = 1 < TH$, $Score_{F_6} = 1.6 > TH$ and $Score_{F_7} = 1.5 > TH$. The information of F_6 and F_7 is finally added to $Sclist$, SC ranks F_6 and F_7 in $Sclist$ according to $Score$, then F_7 is eliminated.

5.2.4. The Subsequent Work

In this section, we focus on verification, decryption of files, and updating the index tree.

A. Verification

$Verify(C_i, MAC_{C_i}, K_2)$: Suppose k encrypted identifiers $FID = \{FID_1, FID_2 \dots, FID_k\}$ are returned in a search result, DU first determines whether FID from CS is the same as FID from SC . Then, the MAC_{C_i} of the file is calculated locally by k_2 to determine whether the CS has tampered with search information. SC sends $\$search$ to DO , sends $\$search_price * \gas_price to the staff member who executed the transaction, and refunds the transaction deposit from DU .

B. Decryption

$Dec(SK_u, C_{K_i}, C_{w_q})$: After getting the key ciphertext C_{K_i} , the user checks whether the SK_u matches with the ciphertext access policy, if it matches, DU can get $A = e(g, g)^{rs}$ and recover the K_i , otherwise, DU has no access to the file. The decryption of K_i is shown in the Decryption Correctness section.

C. Index tree update

$Update(Uptype, I_{T_c})$: The updating of the index tree is the process of replacing the old subtree with the new encrypted one. DO retains the unencrypted index tree locally and denotes T_c as the set of nodes that may be changed during the update process. As an example, in Figure 4, if the file F_1 is deleted in an update operation, then $T_c = \{L_{11}, L_{21}, Root, F_1\}$.

When $Uptype = delete$, delete the leaf node F_i and recompute the other node index vectors in T_c , encrypt node index vectors in T_c by KS . If the operation destroys the balance of the tree, replace it with a pseudo node, then a new indexed subtree I_{T_c} is generated. DO sends I_{T_c} and $Uptype = delete$ to SC , which replaces the original indexed subtree with the help of I_{T_c} . When $Uptype = add$, the leaf node F_i needs to store the key ciphertext C_{K_i} , the file ciphertext message authentication code MAC_{F_i} , encrypted identifier FID_{F_i} and the other operations are the same as the deletion operation. The index update process will not destroy the balance of the tree because the new nodes will be added to the pseudo nodes.

6. Program Analysis

We analyze the correctness and security of the scheme in this section.

6.1. Correctness Analysis

6.1.1. Fuzzy Search Correctness

We denote $I_i = (I_i', I_i'')$ and trapdoor $T = (T', T'')$ for the multi-keyword set w_q . Assuming n file indexes are matched with T in the search, $Score$ is the inner product of I_i and T , denote each element in $Score$ as $P_{x,y}$, which $Score = I_i' \cdot T' + I_i'' \cdot T'' = M_1^T \cdot B_i' \cdot M_1^{-1} \cdot B_i' + M_2^T \cdot B_i'' \cdot M_2^{-1} \cdot B_i''$, $P = \{P_{x,y} | 1 \leq x \leq n, 1 \leq y \leq n\}$ and satisfy (1).

$$\begin{cases} r_{x,y} = b'_{x,y} \cdot t'_y + b''_{x,y} \cdot t''_y = b_{x,y} \cdot (t_y/2 + r') + b_{x,y} \cdot (t_y/2 - r') = b_{x,y} \cdot t_y, s_y = 1 \\ r_{x,y} = b'_{x,y} \cdot t'_y + b''_{x,y} \cdot t''_y = b_{x,y} \cdot (t_y/2 + r') + b_{x,y} \cdot (t_y/2 - r') = b_{x,y} \cdot t_y, s_y = 0 \end{cases} \quad (1)$$

From the above calculations, it can be concluded that the random numbers r and r' are not linked to the result, so the correlation scores of the fuzzy search when the trapdoor is matched with the index inner product are not affected by the random numbers.

6.1.2. Decryption Correctness

After DU receives the key ciphertext C_{k_1} containing the access structure, DU uses attribute private key S_u to determine whether the access structure is satisfied or not, and only if the access structure is satisfied can K_i be recovered, the process is in (2).

$$K_i = \frac{\bar{C}}{\frac{e(C,E)}{A}} = \frac{K_i e(g, g)^{\alpha s}}{\frac{e(h^s, g^{\frac{\alpha+r}{p}})}{e(g, g)^{rs}}} = \frac{K_i e(g, g)^{\alpha s} e(g, g)^{rs}}{e(g, g)^{s(\alpha+r)}} \tag{2}$$

6.2. Security Analysis

Theorem 1. *The scheme is secure under the adaptive selection of keyword attacks when CS and external adversaries can only divulge what they are allowed to.*

Proof. For adversary \mathcal{A} , a stateful simulator \mathcal{B} satisfies the condition $|Pr[Real_{\mathcal{A}}(\lambda)] - Pr[Ideal_{\mathcal{A}, \mathcal{B}}(\lambda)]| \leq negl^*(\lambda)$, where $negl^*(\lambda)$ is a negligible probability. From the proof equivalence in the scheme in [27], the simulation-based game proofs are equivalent to indistinguishable game proofs. \mathcal{A} only needs to analyze the differentiation between \mathcal{B} to win the game, $Pr[Ind_{\mathcal{A}}(\lambda) = 1] - 1/2 \leq negl^*(\lambda)$ is used to prove the scheme is secure. \mathcal{B} simulates the process of generating ciphertexts, indexes, and trapdoors as follows.

- (1) Generate ciphertext C^* by simulation.

According to $\mathcal{L}_1(D) = \left\{ |D|, n, \{ |F_i|, ind(F_i) \}_{i \in [1, n]} \right\}$, \mathcal{B} generates n ciphertexts of $|F_i|_{i \in [1, n]}$ - bit uniformly at random to simulate the encrypted file set $C^* = \{C_1, C_2, \dots, C_n\}$, which is encrypted by AES. According to the semantic security of symmetric encryption, the ciphertext C generated by $Rel_{\mathcal{A}}(\lambda)$ and the ciphertext C^* generated by $Ideal_{\mathcal{A}, \mathcal{B}}(\lambda)$ are computationally indistinguishable, i.e., the advantage of the adversary's victory can be ignored, i.e., $|Pr[Encrypt(F, K_i) \rightarrow C] - Pr[Sim \rightarrow C^*]| \leq negl_1^*(\lambda)$.

- (2) Generate search trapdoor T^* by simulation.

According to $\mathcal{L}_2(D, W_q^*) = \{P(W_q^*), T^*\}$, \mathcal{B} maps each keyword in $W_q^* = (w_1, w_2, \dots, w_q)$ to the bloom filter, its search trapdoor is denoted as $T_Q = (T_{Q'}, T_{Q''})$, where $T_{Q'} = (M_1^{-1} \cdot B_{Q'})$, $T_{Q''} = (M_2^{-1} \cdot B_{Q''})$, $B_{Q'} = (t'_1, t'_2, \dots, t'_m)$, $t'_j = t'_j/2 + r'$, and $t''_j = t''_j/2 - r'$. The probability of \mathcal{B} simulating the search token T_Q is negligible when the KS is unknown. Due to the introduction of the random value r' in the encrypted query and the pseudo-random nature of the key set KS , the simulator \mathcal{B} randomly selects KS^* and r^* to construct the feasible search token with probability $Pr[T_Q = T^*] \approx Pr[r' = r^*] \approx Pr[KS = KS^*] \approx 1/2^k \leq negl_2^*(\lambda)$, thus the advantage of \mathcal{B} to simulate the feasible search trapdoor is negligible.

- (3) Simulated construction of index I^* .

In the process of generating the secure index based on the leakage function $\mathcal{L}_1, \mathcal{L}_2$, simulator \mathcal{B} picks a random value to replace the random number used in the real case. According to the secure KNN encryption algorithm, it is known that the secure KNN algorithm using random numbers is indistinguishable. Meanwhile, the matrix picked for the key set KS is a sufficiently large invertible matrix and KS is not artificially compromised. Thus, the advantage of winning for the polynomial adversary satisfies (3), which is negligible.

$$|Pr[BuildIndex(SK, F_i, KW_{F_i}) \rightarrow I_{real}] - Pr[Sim \rightarrow I^*]| \leq negl_3^*(\lambda) \tag{3}$$

Since the adversary wins this indistinguishability game by analyzing ciphertexts, indexes and search credentials, the advantages of the adversary in distinguishing between real ciphertexts and simulated ciphertexts, real indexes and simulated ciphertexts, and

real search tokens and simulated search tokens, respectively, are denoted as $Adv_{\mathcal{A}}(C, C^*)$, $Adv_{\mathcal{A}}(I_{real}, I^*)$ and $Adv_{\mathcal{A}}(T_Q, T^*)$. Then we get (4).

$$\begin{aligned}
 Pr[Ind_{\mathcal{A}}(\lambda) = 1] &= \frac{1}{2} + Adv_{\mathcal{A}}(C, C^*) + Adv_{\mathcal{A}}(I_{real}, I^*) + Adv_{\mathcal{A}}(T_Q, T^*) \\
 &= \frac{1}{2} + |Pr[Encrypt(F_i, K_i) \rightarrow C] - Pr[Sim \rightarrow C^*]| \\
 &\quad + |Pr[BuildIndex(KS, F_i, KW_{F_i}) \rightarrow I_{real}] - Pr[Sim \rightarrow I^*]| \\
 &\quad + |Pr[Trapdoor(KS, W_q^*) \rightarrow T_Q] - Pr[Sim \rightarrow T^*]| \\
 &\leq \frac{1}{2} + negl_1^*(\lambda) + negl_2^*(\lambda) + negl_3^*(\lambda)
 \end{aligned} \tag{4}$$

In summary, the outputs of $Rel_{\mathcal{A}}(\lambda)$ and $Ideal_{\mathcal{A}, \mathcal{B}}(\lambda)$ obtained for any polynomial adversary are indistinguishable. \square

Theorem 2. *If the blockchain is tamper-proof, the scheme ensures that user transactions are fair.*

Proof. If the CS is dishonest, it will return incorrect file numbers or falsify search results. According to the fair-trade rules set by SC, CS will not receive service fees and will be penalized with loss of deposit. If CS enforces the contract rules honestly, DU gets the correct result and CS gets the corresponding service fee. In addition, the transaction specifies a limit time L_t , when the transaction time is greater than L_t , the user's deposit will be refunded, and the transaction will be ended. \square

Theorem 3. *For malicious cloud servers and other external adversaries, no information can be learned about plaintext files except for ciphertext files.*

Proof. In the scheme, the file is encrypted with the corresponding symmetric key K_i before it is uploaded to the CS, then DO encrypts K_i as C_{K_i} based on CP – ABE and stores it in the index tree $Index_{tree}$. Even though the CS and external adversaries eavesdrop on the ciphertext files in the channel, the difficulty of trying to get the plaintext information is equal to the difficulty of decrypting K_i . Since $\bar{C} = k_i e(g, g)^{as}$ in C_{K_i} , $e(g, g)^{as}$ must be computed to decrypt K_i . Under the discrete logarithm problem, the external adversary and CS to compute $e(g, g)^{as}$ from $C = h^s$ and $e(g, g)^{\alpha}$ is difficult. Based on the *Diffe – Hellman* computational problem, the adversary cannot use the Lagrange interpolation formula to recursively compute $e(g, g)^{rs}$ without the attribute key SK_u that conforms to the access policy and obtain K_i . K_i and the plaintext file can be decrypted when SK_u satisfies the access structure. Since SK_u is privately stored by the DU, it is not possible for CS and external adversaries to learn any plaintext information other than the ciphertext file. \square

7. Efficiency Analysis

The scheme will be analyzed in terms of function, complexity, and performance experimentation in this section.

7.1. Functionality Comparison

The functions of the proposed scheme with the related schemes [15,21,25,28,29] are completed in this section. As shown in Table 2, where the symbol “✓” indicates that the corresponding functionality is satisfied and “✗” indicates that it is not satisfied.

Both the scheme in [15] and the scheme in [29] are about multi-keyword fuzzy searchable encryption. The scheme in [15] realizes dynamic updating by constructing a tree structure but does not support verification of the search result. The scheme in [29] only considers the problem of ciphertext result correctness and does not support dynamic updating. The scheme in [25] utilizes the blockchain mechanism to protect the index, but it does not support fuzzy search and relevance sorting of files, and it cannot maximize the return of the set of files that the user is interested in. The scheme in [28] utilizes authentication and homomorphic encryption to detect misbehavior occurring in cloud servers, and the search results are outsourced to edge computing servers, which can verify the correctness and

integrity, but it would impose additional overhead. Although the scheme [21] supports all the functions, the scheme does not indicate how the process of dynamic updating takes place. It also does not take measures against the possible malicious behavior of the cloud server and does not effectively limit the privileges of the cloud server.

Table 2. Functionality Comparison of Different Schemes.

Scheme	Scheme [15]	Scheme [28]	Scheme [25]	Scheme [29]	Scheme [21]	Ours
Multi-Keyword	✓	✓	✓	✓	✓	✓
Dynamic Updates	✓	×	✓	×	✓	✓
Fuzzy Search	✓	×	×	✓	✓	✓
Integrity Verification	×	✓	✓	×	✓	✓
Correctness Verification	×	✓	✓	✓	✓	✓
Related Sorting	✓	✓	×	✓	✓	✓

The proposed scheme makes up for the shortcomings of the above schemes, and it not only realizes fuzzy search, search result relevance sorting, and dynamic updating but also supports fine-grained access control, search result correctness, and integrity verification. At the same time, smart contracts are used to constrain malicious behaviors that may occur in the search process of cloud servers and protect the rights and interests of users.

7.2. Complexity Theoretical Analysis

In this section, the performance will be analyzed from three aspects: index tree generation, trapdoor generation, and search matching.

- Index tree generation.** The cost of file index generation includes the generation of a plaintext index vector for each file and the encryption of the plaintext index vector. Where the generation of the index is mainly linked to the number of keywords, the encryption of the plaintext vector mainly consists of the factorization operation of the vector and multiplication with a matrix of order $l \times l$. The time overhead of single file index generation is $O(l^2)$. The $Index_{tree}$ generates $(n - 1)$ additional nodes at the internal nodes when the number of files is n . Therefore, the cost of the $Index_{tree}$ generation is a little higher than the other schemes, but it is generated by the DU locally and upload to the smart contract only once.
- Trapdoor Generation.** The cost of trapdoor generation mainly consists of generating an l bit plaintext trapdoor and performing matrix multiplication operations on the plaintext trapdoor, so the cost of trapdoor generation is $O(l^2)$.
- Search Match.** The Search and Match includes traversing the $Index_{tree}$, the calculation of the inner product of trapdoors and index vectors, and sorting the files to filter out rejections based on relevance. In this scheme, only the n_k most relevant ciphertext files are searched. These files and child nodes share the same access path, so the traversal of the $Index_{tree}$ only touches a portion of n nodes i.e., on $r(r \ll n)$ paths from the root to the leaf nodes instead of traversing the child nodes repeatedly, hence the time complexity is $O(r \log n)$.

In Table 3, the efficiency is compared in terms of index generation, trapdoor generation, and search time. Here, n is the number of files, Q is the number of exact keywords, Z is the maximum number of fuzzy keywords, and l denotes the index length in the proposed scheme. In the scheme, l can be treated as a constant and $n \ll Q \ll Z$.

Table 3. Time Complexity of Different Schemes.

Scheme	Scheme [30]	Scheme [31]	Scheme [21]	Ours
Index Generation	$O(ZQ)$	$O(ZQ)$	$O(nl^2)$	$O((2n - 1)l^2)$
Trapdoor Generation	$O(Z)$	$O(Z)$	$O(l^2)$	$O(l^2)$
Search Cost	$O(Z)$	$O(Z \log Q)$	$O(nl)$	$O(r \log n)$

7.3. Experimental Performance

In this section, simulation experiments are conducted. The performance of the scheme is evaluated using Java language in the Windows 10 Home Chinese version. The hardware system is a 64-bit Windows operating system Intel(R) Core (TM) i7-8750H CPU @2.20 GHz with 16 GB of RAM and the experimental dataset is RFC (Request For Comments). Smart contracts are deployed through the solidity language, running in an Ethereum virtual machine. The blockchain implementation is based on the official Ethereum test network Rinkeby and the Ethereum wallet MetaMask is used to function as an account. The numbers of LSH are chosen as $k = 30$, the constructed index and trapdoor vector is $l = 8000$, the file is encrypted by AES-256, and the number of keywords in a file is 80–113.

The scheme excludes keywords with little semantic distinction such as the, where, and a. Simulation experiments and comparisons of our scheme with scheme [15], scheme [28], and scheme [29] are conducted in terms of file index generation, search matching time, and search accuracy.

- Individual File Index Generation and Encryption.** In Figure 5a, the cost of generating a plaintext index vector for a single file index is gradually increasing with the number of file keywords. Meanwhile, the encryption time of the plaintext index vector is constant, which fluctuates slightly at 40 ms.

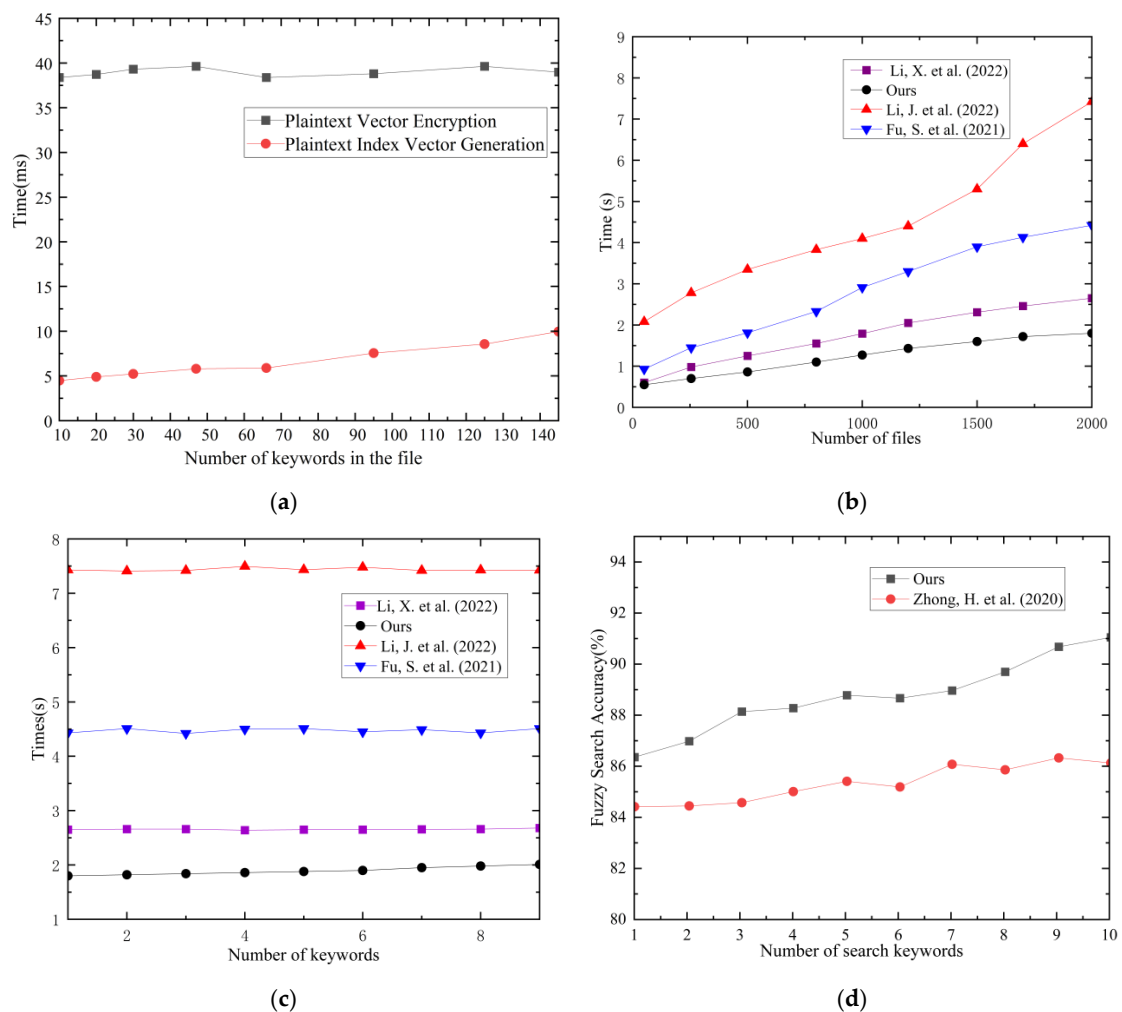


Figure 5. (a) Index generation time. (b) The search matching time when search keyword number is 3. (c) The search matching time when the file number is 2000. (d) The fuzzy search accuracy. Zhong, H. et al (2020) refers to the scheme in ref. [15], Li, X. et al (2022) refers to the scheme in ref. [21], Li, J. et al (2022) refers to the scheme in ref. [28] and Fu, S. et al (2021) refers to the scheme in ref. [29].

- **Search Match.** In Figure 5b, as the number of files gradually increases, all 4 schemes basically show an increasing trend. When the number of files is no more than 1000, all the schemes increase at a relatively small level. The time cost and growth rate required for searching in the proposed scheme are lower than those of the schemes in [25,29], and as the number of files increases, the scheme follows up with a slight increase compared to the beginning. Although the search time of the scheme in [21] also consistently increases at a lower level, the search time of our scheme is still superior to it. When the number of files is 2000, the search times of the schemes in [28,29] are 7.42 s and 4.42 s, respectively, while the time cost of our scheme is only 2.21 s. In Figure 5c, the search matching costs of the schemes in [21,28,29] tend to stabilize, while the time cost in our scheme shows a small linear increase. When the number of search keywords is 9, the time costs of the scheme in [28,29], and [21] are 7.42 s, 4.51 s, and 2.65 s, respectively, while the time cost of our scheme is only about 1.85 s.
- **Fuzzy Search Accuracy.** Since our scheme and the scheme in [15] have similarities in index construction which uses *unigram* to transform keywords, the scheme in [15] is chosen to compare fuzzy search accuracy. The fuzzy keywords are constructed by replacing one letter in the keyword with another letter randomly, and the check accuracy is used to test the search results. The accuracy test method $\Delta P = p' / p$ from the scheme in [32] is used in our scheme, where p indicates the number of search files returned and p' indicates the number of real files returned.

As seen in Figure 5d, the search accuracy increases gradually with the increase in the number of search keywords, which makes the scheme more capable of distinguishing the files that satisfy the user's search conditions. The search accuracy of the scheme in [15] is around 85%, while the search accuracy of our scheme can reach 91% with the increase in valid search keywords.

8. Conclusions

In this paper, an effective semi-decentralized multi-keyword fuzzy searchable encryption scheme is proposed. We improved the index tree storage to support fine-grained access control without losing search efficiency. In addition, CP – ABE is introduced into the scheme to support multi-user scenarios. Experimental results show that our scheme has certain advantages in efficiency compared to similar schemes. In a word, the proposed scheme is rich with additional functions such as verification, updating, etc., while at the same time presenting a high performance.

Author Contributions: Conceptualization, X.Y. and P.C.; methodology, X.Y. and P.C.; software, X.Y.; validation, X.Y. and P.C.; formal analysis, X.Y., P.C. and J.Z.; investigation, P.C.; resources, X.Y. and Y.T.; data curation, P.C.; writing—original draft preparation, X.Y. and P.C.; writing—review and editing, X.Y. and P.C.; visualization, X.Y. and P.C.; supervision, X.Y. and Y.T.; project administration, X.Y. and Y.T.; funding acquisition, X.Y. and Y.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by [Programs for Science and Technology Development of Henan Province] grant number [242102210152], [the Fundamental Research Funds for the Universities of Henan Province] grant number [NSFRF240620], [Key Scientific Research Project of Henan Higher Education Institutions] grant number [24A520015] and [National Natural Science Foundation of China] grant number [62472144]. And The APC was funded by [National Natural Science Foundation of China] grant number [62472144].

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Song, D.X.; Wagner, D.; Perrig, A. Practical techniques for searches on encrypted data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, S&P 2000, Berkeley, CA, USA, 14–17 May 2000; pp. 44–55.
2. Xu, G.; Li, H.; Ren, H.; Lin, X.; Shen, X. DNA Similarity Search With Access Control Over Encrypted Cloud Data. *IEEE Trans. Cloud Comput.* **2022**, *10*, 1233–1252. [[CrossRef](#)]
3. Tong, Q.; Miao, Y.; Liu, X.; Choo, K.-K.R.; Deng, R.H.; Li, H. VPSL: Verifiable Privacy-Preserving Data Search for Cloud-Assisted Internet of Things. *IEEE Trans. Cloud Comput.* **2022**, *10*, 2964–2976. [[CrossRef](#)]
4. Boneh, D.; Waters, B. Conjunctive, Subset, and Range Queries on Encrypted Data. In *Theory of Cryptography*; Vadhan, S.P., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4392, pp. 535–554. [[CrossRef](#)]
5. Chen, L.; Cong, G.; Jensen, C.S.; Wu, D. Spatial Keyword Query Processing: An Experimental Evaluation. In Proceedings of the VLDB Endowment, Riva del Garda, Trento, Italy, 26–30 August 2013.
6. Tong, Q.; Li, X.; Miao, Y.; Liu, X.; Weng, J.; Deng, R. Privacy-Preserving Boolean Range Query with Temporal Access Control in Mobile Computing. *IEEE Trans. Knowl. Data Eng.* **2022**, *35*, 5159–5172. [[CrossRef](#)]
7. Li, J.; Wang, Q.; Wang, C.; Cao, N.; Ren, K.; Lou, W. Fuzzy Keyword Search over Encrypted Data in Cloud Computing. In Proceedings of the 2010 Proceedings IEEE INFOCOM, San Diego, CA, USA, 14–19 March 2010; pp. 1–5. [[CrossRef](#)]
8. Chuah, M.; Hu, W. Privacy-Aware BedTree Based Solution for Fuzzy Multi-keyword Search over Encrypted Data. In Proceedings of the 2011 31st International Conference on Distributed Computing Systems Workshops, Minneapolis, MN, USA, 20–24 June 2011; pp. 273–281. [[CrossRef](#)]
9. Wang, B.; Yu, S.; Lou, W.; Hou, Y.T. Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, 26 April–1 May 2015; pp. 2092–2100.
10. Chen, J.; He, K.; Deng, L.; Yuan, Q.; Du, R.; Xiang, Y. EliMFS: Achieving Efficient, Leakage-Resilient, and Multi-Keyword Fuzzy Search on Encrypted Cloud Data. *IEEE Trans. Serv. Comput.* **2020**, *13*, 1072–1085. [[CrossRef](#)]
11. Shan, B.; Yao, Y.; Li, W.; Zuo, X.; Yu, N. Fuzzy Keyword Search over Encrypted Cloud Data with Dynamic Fine-grained Access Control. In Proceedings of the 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Wuhan, China, 9–11 December 2022; pp. 1340–1347. [[CrossRef](#)]
12. Xia, Z.; Wang, X.; Sun, X.; Wang, Q. A Secure and Dynamic Multi-Keyword Ranked Search Scheme over Encrypted Cloud Data. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 340–352. [[CrossRef](#)]
13. Fu, Z.; Wu, X.; Wang, Q.; Ren, K. Enabling Central Keyword-Based Semantic Extension Search Over Encrypted Outsourced Data. *IEEE Trans. Inform. Forensics Secur.* **2017**, *12*, 2986–2997. [[CrossRef](#)]
14. Fu, Z.; Wu, X.; Guan, C.; Sun, X.; Ren, K. Toward Efficient Multi-Keyword Fuzzy Search Over Encrypted Outsourced Data With Accuracy Improvement. *IEEE Trans. Inform. Forensic Secur.* **2016**, *11*, 2706–2716. [[CrossRef](#)]
15. Zhong, H.; Li, Z.; Cui, J.; Sun, Y.; Liu, L. Efficient dynamic multi-keyword fuzzy search over encrypted cloud data. *J. Netw. Comput. Appl.* **2020**, *149*, 102469. [[CrossRef](#)]
16. Wang, C.; Ren, K.; Yu, S.; Urs, K.M.R. Achieving usable and privacy-assured similarity search over outsourced cloud data. In Proceedings of the 2012 Proceedings IEEE INFOCOM, Orlando, FL, USA, 25–30 March 2012; pp. 451–459. [[CrossRef](#)]
17. Zhang, H.; Zhao, S.; Guo, Z.; Wen, Q.; Li, W.; Gao, F. Scalable Fuzzy Keyword Ranked Search Over Encrypted Data on Hybrid Clouds. *IEEE Trans. Cloud Comput.* **2023**, *11*, 308–323. [[CrossRef](#)]
18. Kuzu, M.; Islam, M.S.; Kantarcioglu, M. Efficient Similarity Search over Encrypted Data. In Proceedings of the 2012 IEEE 28th International Conference on Data Engineering, Arlington, VA, USA, 1–5 April 2012; pp. 1156–1167. [[CrossRef](#)]
19. Wang, B.; Yu, S.; Lou, W.; Hou, Y.T. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In Proceedings of the IEEE INFOCOM 2014—IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; pp. 2112–2120. [[CrossRef](#)]
20. Tong, Q.; Miao, Y.; Weng, J.; Liu, X.; Choo, K.-K.R.; Deng, R. Verifiable Fuzzy Multi-keyword Search over Encrypted Data with Adaptive Security. *IEEE Trans. Knowl. Data Eng.* **2022**, *35*, 5386–5399. [[CrossRef](#)]
21. Li, X.; Tong, Q.; Zhao, J.; Miao, Y.; Ma, S.; Weng, J. VRFMS: Verifiable Ranked Fuzzy Multi-keyword Search over Encrypted Data. *IEEE Trans. Serv. Comput.* **2022**, *16*, 698–710. [[CrossRef](#)]
22. Cai, C.; Weng, J.; Yuan, X.; Wang, C. Enabling Reliable Keyword Search in Encrypted Decentralized Storage with Fairness. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 131–144. [[CrossRef](#)]
23. Wang, S.; Zhang, Y.; Zhang, Y. A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems. *IEEE Access* **2018**, *6*, 38437–38450. [[CrossRef](#)]
24. Hu, S.; Cai, C.; Wang, Q.; Wang, C.; Luo, X.; Ren, K. Searching an Encrypted Cloud Meets Blockchain: A Decentralized, Reliable and Fair Realization. In Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications, Honolulu, HI, USA, 15–19 April 2018; pp. 792–800. [[CrossRef](#)]
25. Chen, Z.; Wu, A.; Li, Y.; Xing, Q.; Geng, S. Blockchain-Enabled Public Key Encryption with Multi-Keyword Search in Cloud Computing. *Secur. Commun. Netw.* **2021**, *2021*, 6619689. [[CrossRef](#)]
26. Guo, Y.; Zhang, C.; Wang, C.; Jia, X. Towards Public Verifiable and Forward-Privacy Encrypted Search by Using Blockchain. *IEEE Trans. Dependable Secur. Comput.* **2022**, *20*, 2111–2126. [[CrossRef](#)]

27. Chai, Q.; Gong, G. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In Proceedings of the 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, Canada, 10–15 June 2012; pp. 917–922. [[CrossRef](#)]
28. Li, J.; Ma, J.; Miao, Y.; Chen, L.; Wang, Y.; Liu, X. Verifiable Semantic-Aware Ranked Keyword Search in Cloud-Assisted Edge Computing. *IEEE Trans. Serv. Comput.* **2022**, *15*, 3591–3605. [[CrossRef](#)]
29. Fu, S.; Zhang, Q.; Jia, N.; Xu, M. A Privacy-preserving Fuzzy Search Scheme Supporting Logic Query over Encrypted Cloud Data. *Mob. Netw. Appl.* **2021**, *26*, 1574–1585. [[CrossRef](#)]
30. Ge, X.; Yu, J.; Hu, C.; Zhang, H.; Hao, R. Enabling Efficient Verifiable Fuzzy Keyword Search Over Encrypted Data in Cloud Computing. *IEEE Access* **2018**, *6*, 45725–45739. [[CrossRef](#)]
31. Huang, R.; Li, Z.; Wu, G. A Verifiable Encryption Scheme Supporting Fuzzy Search. In *Security, Privacy, and Anonymity in Computation, Communication, and Storage*; Wang, G., Feng, J., Bhuiyan, M.Z.A., Lu, R., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2019; Volume 11611, pp. 397–411. [[CrossRef](#)]
32. Xiangyang, Z.; Hua, D.; Xun, Y.; Geng, Y.; Xiao, L. MUSE: An Efficient and Accurate Verifiable Privacy-Preserving Multikeyword Text Search over Encrypted Cloud Data. *Secur. Commun. Netw.* **2017**, *2017*, 1923476. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.