Article

A Key-Policy Searchable Attribute-Based Encryption Scheme for Efficient Keyword Search and Fine-Grained Access Control over Encrypted Data

Hui Yin ¹ , Yinqiao Xiong ^{1,2,*} , Jixin Zhang ^{3,*}, Lu Ou ³ , Shaolin Liao ^{4,5} and Zheng Qin ³

¹ College of Computer Engineering and Applied Mathematics, Changsha University, Changsha 410022, China; yhui@ccsu.edu.cn

² College of Computer, National University of Defense Technology, Changsha 410073, China

³ College of Information Science and Engineering, Hunan University, Changsha 410082, China; oulu9676@gmail.com (L.O.); zqin@hnu.edu.cn (Z.Q.)

⁴ Argonne National Laboratory, Cass Avenue, Lemont, IL 60439, USA; sliao@anl.gov

⁵ Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616, USA

* Correspondence: yq.xiong@ccsu.edu.cn (Y.X.); zhangjixin@hnu.edu.cn (J.Z.); Tel.: +86-133-9760-8869 (Y.X.)

Received: 31 January 2019; Accepted: 25 February 2019; Published: 28 February 2019

Abstract: Attribute based encryption is a promising technique that achieves flexible and fine-grained data access control over encrypted data, which is very suitable for a secure data sharing environment such as the currently popular cloud computing. However, traditional attribute based encryption fails to provide an efficient keyword based search on encrypted data, which somewhat weakens the power of this encryption technique, as search is usually the most important approach to quickly obtain data of interest from large-scale dataset. To address this problem, attribute based encryption with keyword search (ABKS) is designed to achieve fine-grained data access control and keyword based search, simultaneously, by an ingenious combination of attribute based encryption and searchable encryption. Recently, several ABKS schemes have been constructed in secure cloud storage system for data access control and keyword search. Nonetheless, each of these schemes has some defects such as impractical computation overhead and insufficient access policy expression. To overcome these limitations, in this paper, we design a Key-Policy Searchable Attribute-based Encryption Scheme (KPSABES) based on the full-blown key-policy attribute-based encryption proposed by Vipul Goyal et al. By novel design, our scheme not only inherits all advantages of that scheme but also achieves efficient and secure keyword search over encrypted data. We provide the detailed performance analyses and security proofs for our scheme. Extensive experiments demonstrated that our proposed scheme is superior in many aspects to the similar work.

Keywords: access control; cloud computing; key-policy attribute-based encryption; searchable encryption

1. Introduction

1.1. Motivation

The emerging cloud computing paradigm motivates organizations and individuals to outsource their private data to the cloud server for saving local IT construction and maintenance cost. Regardless of numerous advantages of cloud computing such as multi-tenant, rapid deployment, reliability, flexibility, etc., data security and privacy is still a knotty problem to hinder the wide adoption of cloud computing, since the data are out of the control of data owners once being outsourced to the remote cloud center [1]. An intuitive way to protect data is encryption [2,3]. However, traditional encryption

techniques block the fundamental utilizations of data such as search, calculation, and so on. Without doubt, a data user can download all ciphertexts from the cloud server and decrypt them to obtain interested data. Obviously, this straightforward method will incur impractical communication and computation overhead in the cloud computing environment, where the data are *big*.

We were first motivated by the full-blown key-policy attribute-based encryption scheme (KP-ABE) and its application example proposed by Vipul Goyal et al. [4]. Essentially, the scheme in [4] further develops the first attribute-based encryption scheme by allowing for a wide variety of access structures, not only a fixed threshold gate. Therefore, the more flexible and fine-grained data access control over encrypted data can be achieved in a secure data sharing setting. In brief, in their construction, each data user's key is associated with a specified tree-access structure supporting AND, OR, and threshold gates and a message is encrypted under a set of attributes. A user's key can recover the message from the corresponding ciphertext if and only if the attributes in the ciphertext satisfy the access structure in the key. In addition, the authors demonstrated an important application of their KP-ABE scheme in the secure forensic analysis system, where all audit logs are encrypted by their proposed KP-ABE scheme. More specifically, in that system, a forensic analyst is issued a secret key associated with a specified access structure and the audit log records are encrypted under a set of attributes. As a result, the forensic analyst can even download all ciphertexts but can only decrypt those encrypted audit log records whose attributes satisfy the access structure in the forensic analyst's key. Thus, the access control over encrypted data can be achieved in a fine-grained manner. Nevertheless, in the practical work, the forensic analyst may only want to obtain some audit logs of interested. If the forensic analyst can first retrieve the encrypted audit logs that are within his/her access privileges and that are of interest and then decrypt them, the communication cost and decryption time could be greatly reduced. Motivated by this practical requirement, in this paper, we design a key-policy searchable attribute-based encryption scheme, named KPSABES, based on Goyal et al.'s key-policy attribute-based encryption scheme [4]. According to our design, KPSABES can achieve efficient keyword-based search and fine-grained access control over encrypted data, simultaneously. Next, to further illustrate our motivation, we review the related works on the following three topics: searchable encryption, attribute-based encryption, and searchable attribute-based encryption.

1.2. Related Works

1.2.1. Searchable Encryption

Searchable encryption (SE) allows a server to perform search over encrypted data according to a search token submitted by a data user. Song et al. proposed the first practical SE scheme in [5] with linear search complexity. Curtmola et al. [6] used inverted index structure to construct a more efficient SE construction that achieves sub-linear search complexity. Kamara et al. [7] introduced the dynamic SE technique. A dynamic SE scheme supports secure and efficient file updating. Recently, the dynamic SE schemes are further investigated in [8–10] to achieve better efficiency and security. To allow for the correctness and completeness verification of search results, the verifiable SE was introduced by Kurosawa and Ohtaki [11] against the unreliable server [12], which may return incorrect and incomplete search results to the data user. By using the emerging blockchain technique, decentralized SE schemes are designed in [13,14], which can ensure that the data user can always receive the correct and complete search results. Cloud computing promotes the development of data outsourcing services, and the SE schemes over encrypted outsourced cloud data are adequately researched in [15–21]. These works mainly focus on the multi-keyword ranked search, fuzzy search, personalized search, etc. Since these above schemes are implemented in the symmetric cryptosystem, they are referred as symmetric searchable encryption (SSE). The first public-key based SE scheme was proposed by Boneh et al. [22]. Later, the keyword based conjunctive search and range search are also explored in the public-key setting in [23–26]. However, these traditional searchable encryption schemes do not consider search permissions of data users and cannot achieve flexible access control on encrypted data.

1.2.2. Attribute Based Encryption

Attribute-based encryption (ABE) is a novel cryptographic primitive that can enforce fine-grained data access control via cryptographic means. Sahai and Water designed the first attribute-based encryption, called Fuzzy Identity-Based Encryption (FIBE) [27]. In FIBE, a message is encrypted with a set of attributes ω , and a private key is associated with a set of attributes ω' . If and only if the two attribute sets satisfy condition $|\omega \cap \omega'| \geq d$, the private can decrypt the message, where d is a pre-set fixed threshold. To make access control structure in ABE more expressive and flexible, ABE is further developed in [4,28–34]. These schemes can be generally classified into two categories: key-policy attribute-based encryption (KP-ABE) and ciphertext-policy attribute-based encryption (CP-ABE). In KP-ABE [4,32–34], a message is encrypted with a set of attributes and the user's private key is associated with an access policy. If the attributes in the ciphertext satisfy the access policy in the private key, the key can recover the original message from the ciphertext. On the contrary, in CP-ABE schemes [28–31], a message is encrypted with an access policy and the user's key is associated with his/her attributes. If and only if the user's attributes satisfy the access policy in the ciphertext, the decryption can be successful. Recently, some anonymous attribute-based encryption schemes are designed in [35–38], where the attribute information is hidden for preventing from revealing privacy. Although the above schemes can provide the flexible and fine-grained access control on the encrypted data, they limit data searching.

1.2.3. Searchable Attribute-Based Encryption

To achieve effective keyword-based search and data access control over encrypted data, several searchable attribute-based encryption schemes are proposed [39–48]. These schemes combine searchable encryption and attribute-based encryption techniques to achieve keyword-based search and data access control over encrypted data. Nonetheless, each of these schemes has some limitations. Specifically, the schemes in [39,40] use an access structure that only allows AND policies and the schemes in [41–44,48] consider AND and OR gates of attributes but do not support flexible threshold gate. The authors proposed an efficient ciphertext-policy searchable attribute-based encryption scheme in [45], which supports AND, OR, and threshold gate. However, the query keyword in the scheme is vulnerable against the chosen plaintext attack, since the search token generation algorithm is a deterministic encryption. In addition, the schemes in [46,47] use composite-order group to set up pairing environment, which incurs the impractical search complexity [43].

In this paper, we design a key-policy searchable attribute-based encryption scheme based on Goyal et al.'s scheme [4], which efficiently works in the prime order group and flexibly supports AND, OR, and threshold gates. Our proposed scheme is applicable to any real applications in the encrypted data share setting, where fine-grained data access control and efficient keyword search should be able to achieved just like the audit log application mentioned at the beginning of this section.

It needs to be emphasized that Zheng et al. [49] also constructed a similar key-policy attribute-based keyword search scheme (KP-ABKS) based on the scheme in [4]. However, KP-ABKS has to introduce extra ciphertext components and time-consuming operations on the basis of the scheme [4] to achieve data searching. By skillfully design, our scheme does not need any additional ciphertext components and expensive operations to support data searching. The performance analyses and experimental results further show that our proposed scheme is superior in many aspects to theirs, including the search complexity, which is usually regarded as the core evaluation index for a search system.

1.3. Contributions

In this paper, we make three key contributions:

- (1) We propose KPSABES, a key-policy searchable attribute-based encryption scheme. KPSABES enables efficient keyword based search and flexible and fine-grained access control over encrypted data by leveraging the searchable encryption and the key-policy attribute-based encryption.
- (2) We analyzed the performance complexity for our proposed algorithms in KPSABES and the similar scheme KP-ABKS in [49]. In addition, we provide the thorough security proofs for our proposed scheme.
- (3) We implemented our proposed KPSABES and the similar work KP-ABKS in [49], and made the thorough performance comparisons between these two schemes through extensive experiments. The results demonstrate that KPSABES is superior in many aspects to PK-ABKS.

The rest of this paper is organized as follows. We introduce the background knowledge used to construct KPSABES in Section 2. Section 3 presents the system model and the security model for our proposed problem. We give formal algorithm definition and implementation details in Section 4. The performance analysis and security proof of our scheme are provided in Sections 5 and 6, respectively. We present our experimental evaluation of our scheme in Section 7. Section 8 concludes this paper.

2. Background Knowledge

In this section, we introduce the various background knowledge that is used to construct our proposed KPSABES scheme.

2.1. Bilinear Map

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order p , where g denotes a generator of $\mathbb{G}_1$. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ follows the properties:

- (1) Computability: For all $u, v \in \mathbb{G}_1$, there exists an efficient polynomial time algorithm to compute $e(u, v) \in \mathbb{G}_2$.
- (2) Bilinearity: For all $u, v \in \mathbb{G}_1$ and $x, y \in \mathbb{Z}_p^*$, $e(u^x, v^y) = e(u, v)^{xy}$ holds.
- (3) Non-degeneracy: $e(g, g) \neq 1$, where g is a generator of $\mathbb{G}_1$.

2.2. Access Tree

The access tree is an expressive access structure, which has been widely used in attribute-based encryption. In an access tree denoted by T , each non-leaf node can represent an AND gate, an OR gate, or a threshold gate according to its threshold value. Generally, for any node $x \in T$, we use notations num_x and k_x to denote the number of children and threshold value of x , respectively. For any non-leaf node x , if $k_x = 1$, x represents an OR gate; if $k_x = num_x$, it represents an AND gate; and, if $1 < k_x < num_x$, x is a threshold gate. For any leaf node x , we define $k_x = 1$ and $num_x = 0$.

For ease of description to an access tree, we define the following notations.

- (1) $parent(x)$: The parent of node x .
- (2) $index(x)$: The ordering number of node x with respect to its parent node $parent(x)$. For example, if x is the first child (from left to right) of node $parent(x)$, then $index(x) = 1$, and so forth.
- (3) $attr(x)$: The attribute associated with the leaf node x in an attribute-based encryption system.

To help understand the access tree and related notations, we give a concrete example, as shown in Figure 1, where the number in each node denotes the threshold value of the node. According to our definition, we have $num_R = 3$, $num_{N_1} = 2$, $k_R = 3$, $k_{N_2} = 1$, $N_1 = parent(b)$, $index(b) = 2$, $index(c) = 1$, and $index(g) = 3$. In the access tree, nodes R and N_1 are AND gates, N_2 is an OR gate, and N_3 is a threshold gate.

Satisfying an access tree. Given a set of attributes $\mathbb{A}$ and an access tree T with the root node r , T_x denotes the subtree of T rooted at the node x , and we define

$$\begin{cases} T_x(\mathbb{A}) = 1 & \text{If the attribute set } \mathbb{A} \text{ satisfies } T_x \\ T_x(\mathbb{A}) = 0 & \text{Otherwise} \end{cases} \quad (1)$$

$T_x(\mathbb{A})$ can be recursively computed as follows. If x is a non-leaf node, evaluate $T_{x'}(\mathbb{A})$ for all children x' of node x . $T_x(\mathbb{A})$ returns 1 if and only if at least k_x children return 1. If x is a leaf node, $T_x(\mathbb{A})$ returns 1 if and only if $\text{attr}(x) \in \mathbb{A}$. Thus, according to the above recursive computation, if $\mathbb{A}$ satisfies T , then $T_r(\mathbb{A}) = 1$; otherwise $T_r(\mathbb{A}) = 0$. For instance, the attribute sets $\{aa, ab, ac, ae, ag\}$ and $\{aa, ab, ad, af, ag\}$ satisfy the above access tree, but none of the attribute sets $\{aa, ac, ae\}$, $\{aa, ab, ae, af\}$, and $\{aa, ab, ac, ae\}$ satisfy it, where $\text{attr}(a) = aa, \text{attr}(b) = ab, \text{attr}(c) = ac, \text{attr}(d) = ad, \text{attr}(e) = ae, \text{attr}(f) = af, \text{attr}(g) = ag$.

Figure 1. An example of the access tree.

2.3. Complexity Assumption

- (1) Decisional Diffie–Hellman (DDH) assumption. Let g be a generator of group $\mathbb{G}$ with order p ; choose three elements $a, b, c \in \mathbb{Z}_p^*$ at random uniformly. The DDH assumption is that no probabilistic polynomial-time adversary $\mathcal{A}$ can distinguish the tuple (g, g^a, g^b, g^c) from the tuple (g, g^a, g^b, g^{ab}) with a non-negligible advantage.
- (2) Decisional Bilinear Diffie–Hellman (DBDH) assumption. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative groups of order p , where g is a generator of $\mathbb{G}_1$ and e denotes a bilinear map; choose four elements $a, b, c, z \in \mathbb{Z}_p^*$ at random uniformly. DBDH assumption is that no probabilistic polynomial-time adversary $\mathcal{A}$ can distinguish the tuple (g, g^a, g^b, g^c, g^z) from the tuple $(g, g^a, g^b, g^c, e(g, g)^{abc})$ with a non-negligible advantage.

3. System Model and Security Model

In this section, we present the system model and security model for our KPSABES.

3.1. System Model

The system model of KPSABES consists of three entities: the data owner, the data users, and the server (Figure 2). To ensure the confidentiality, the data owner encrypts the data files using any semantically secure symmetrical encryption scheme such as AES. Meanwhile, the data owner extracts index keywords from data files and defines a set of attributes for each index keywords according to its access permission. Then, each index keyword is encrypted under the corresponding attribute set. Finally, the encrypted data files along with the encrypted index keywords are outsourced to the server (e.g., the cloud server). When a data user wishes to join the system, the data owner first defines an access tree for the data user according to his/her system search permission. Then, the data owner uses the access tree to construct a private key for the data user. Finally, the generated private key along with the symmetrical key encrypting the data files is issued to the data user via secure communication channels. On the other hand, the data user can use the authorized private key to encrypt a query keyword of interest to generate search token, which is then submitted to the server. Upon receiving the search token, the server performs search over encrypted index keywords and returns the search results to the data user, who uses the symmetrical key to decrypt the search results locally. In the whole processes, the server knows nothing about the index keywords and the query keyword. In addition, all files and index keywords are organized as inverted index structure to achieve sub-linear search complexity.

In our system, we introduce attribute based encryption to achieve keyword authorization search in a fine-grained manner. In essence, if and only if the attribute set in an index keyword satisfies the access tree in the data user's key, the data user bears the search permission of the index keyword. For example, in Figure 2, given two index keywords w_i and w_j , which are encrypted by the attribute set {Computer, Professor} and {Math, Doctor} respectively, obviously, the data user has access to the index keyword w_i , but not w_j , where $[w_i]$ and $[w_j]$ denote the ciphertexts of index keyword w_i and w_j .

Figure 2. System model.

3.2. Security Model

Acknowledgedly, the security goal of the searchable encryption is to prevent the adversary from obtaining the underlying keyword information from the encrypted index keyword and the search token. Usually, the server is considered as the “curious but honest” adversary, and both the data owner and data users are trusted [50].

We use the selective security against chosen-keyword attack game and adaptively chosen-plaintext attack game to formalize the security properties of the index keyword and the search token, respectively. We provide the formal security definitions for our scheme through the following games between a challenger $\mathcal{C}$ and a probabilistic polynomial-time adversary $\mathcal{A}$.

Selectively Chosen-Keyword Attack Game:

Setup. The challenger $\mathcal{C}$ generates a public parameter and a master key and sends the public parameter to the adversary $\mathcal{A}$. $\mathcal{A}$ specifies a set of attributes $\mathbb{A}^*$ that he wishes to be challenged upon and submits it to $\mathcal{C}$.

Phase 1. The adversary $\mathcal{A}$ can issue private key queries many times corresponding to access trees $T_1, \dots, T_q$. The only restriction is that none of these access trees are satisfied by the challenge attribute set $\mathbb{A}^*$. $\mathcal{A}$ can use these private keys responded by $\mathcal{C}$ to encrypt query keywords to generate legal search tokens.

Challenge. The adversary $\mathcal{A}$ outputs two keywords w_0 and w_1 and then sends them to $\mathcal{C}$. $\mathcal{C}$ fairly chooses a bit $b \in \{0, 1\}$ and encrypts w_b with the challenge attribute set $\mathbb{A}^*$. The ciphertext is transferred to the adversary.

Phase 2. $\mathcal{A}$ continues to ask the challenger $\mathcal{C}$ for many private keys corresponding to access trees $T_{q+1}, \dots, T_{q+m}$. The same restriction as in Phase 1 has to be satisfied in this phase.

Guess. The adversary $\mathcal{A}$ outputs a guess b' of b .

We define the advantage that the probabilistic polynomial-time adversary $\mathcal{A}$ wins the above game as $|\Pr[b = b'] - \frac{1}{2}|$.

Definition 1. A searchable attribute based encryption scheme is selectively secure against chosen-keyword attack if the above advantage is negligible. In other words, the index keyword encryption construction does not reveal any plaintext information of the keyword under the selectively chosen-keyword attack model.

Adaptively Chosen-Plaintext Attack Game:

Setup. The challenger $\mathcal{C}$ generates a public parameter and a master key and sends the public parameter to the adversary $\mathcal{A}$.

Phase 1. The adversary $\mathcal{A}$ is allowed to adaptively query for the ciphertext of any search keyword. For any queried search keyword $\hat{w}$, $\mathcal{B}$ returns the ciphertext $\text{ST}_{\mathcal{B}}(\hat{w})$ to $\mathcal{A}$.

Challenge. The adversary $\mathcal{A}$ outputs two query keywords w_0 and w_1 and then sends them to $\mathcal{C}$. $\mathcal{C}$ fairly chooses a bit $b \in \{0, 1\}$ and encrypts w_b with the challenge attribute set $\mathbb{A}^*$. The ciphertext is transferred to the adversary.

Phase 2. $\mathcal{A}$ continues to ask the challenger $\mathcal{C}$ for many private keys and encrypts any query keyword w (note that w can be w_0 and w_1) to generate legal search token.

Guess. The adversary $\mathcal{A}$ outputs a guess b' of b .

We define the advantage that the probabilistic polynomial-time adversary $\mathcal{A}$ wins the above game as $|\Pr[b = b'] - \frac{1}{2}|$.

Definition 2. A searchable attribute based encryption scheme is semantically secure against the adaptively chosen-plaintext attack if the above advantage is negligible. In other words, the search token does not reveal any plaintext information of the query keyword under the adaptively chosen-plaintext attack model.

4. Definition and Construction of KPSABES

In this section, we give the definition and describe the construction for our proposed KPSABES.

4.1. Definition

Our proposed KPSABES is composed of the following five polynomial-time algorithms.

- $\text{Setup}(1^\lambda) \rightarrow (\text{PK}, \text{MK})$. The data owner runs setup algorithm to output system public parameter PK and the master private key MK, where λ is a security parameter as the input of this algorithm.
- $\text{Keygen}(\text{PK}, \text{MK}, \mathbb{T}_u) \rightarrow \mathbb{K}_u$. The data owner runs keygen algorithm to generate the private key $\mathbb{K}_u$ for a data user u , where PK and MK are the system public parameter and the master private, respectively, and $\mathbb{T}_u$ is an access tree which defines u 's search permission.
- $\text{Enc}(\text{PK}, w, \mathbb{A}) \rightarrow \mathbb{I}_w$. The data owner runs enc algorithm to encrypt an index keyword w and outputs w 's ciphertext $\mathbb{I}_w$, where PK is the system public parameter, w is an index keyword, and $\mathbb{A}$ is a set of attributes which defines w 's access permission.
- $\text{Searchtoken}(\text{PK}, \mathbb{K}_u, \hat{w}) \rightarrow \text{ST}_u(\hat{w})$. The data user u runs searchtoken algorithm to encrypt a query keyword $\hat{w}$ and outputs $\hat{w}$'s search token $\text{ST}_u(\hat{w})$, where PK is the public parameter, $\mathbb{K}_u$ is the data user u 's private key, and $\hat{w}$ denotes any query keyword.
- $\text{Search}(\text{PK}, \mathbb{I}_w, \mathbb{T}_u(\hat{w})) \rightarrow 1/0$. The server runs search algorithm to perform matching between the encrypted index keyword $\mathbb{I}_w$ and the search token $\mathbb{T}_u(\hat{w})$ with the help of the system public parameter PK. The algorithm outputs 1 if the data user u has access to the index keyword w and w is identical to $\hat{w}$, simultaneously; otherwise, it outputs 0.

Correctness: KPSABES is correct if

$$\Pr \left[\begin{array}{l} \text{Search}(\text{PK}, \mathbb{I}_w, \mathbb{T}_u(\hat{w})) \rightarrow 1/0 \\ \left[\begin{array}{l} \text{Setup}(1^\lambda) \rightarrow (\text{PK}, \text{MK}); \\ \text{Keygen}(\text{PK}, \text{MK}, \mathbb{T}_u) \rightarrow \mathbb{K}_u; \\ \text{Enc}(\text{PK}, w, \mathbb{A}) \rightarrow \mathbb{I}_w; \\ \text{Searchtoken}(\text{PK}, \mathbb{K}_u, \hat{w}) \rightarrow \text{ST}_u(\hat{w}). \end{array} \right] \end{array} \right] = 1 \quad (2)$$

4.2. Construction

In this subsection, we formally design our KPSABES and provide the concrete implementation of each algorithm of KPSABES.

4.2.1. Setup

When inputting an enough security parameter, the Setup algorithm establishes the running environment of KPSABES and generates the system public parameter and the master key. More specifically, the algorithm first generates two cycle multiplication groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p , and creates a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ and a hash function $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p^*$. Let g denote a generator of $\mathbb{G}_1$. Then, the algorithm chooses a random value $\alpha \in \mathbb{Z}_p^*$ and sets $g_1 = g^\alpha$ and further computes $e(g_1, g)$. Define the Lagrange coefficient

$$\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x - j}{i - j}, \quad (3)$$

where S denotes a set of elements in $\mathbb{Z}_p^*$ and $i, j \in \mathbb{Z}_p^*$. In addition, let n be the maximum size of the attribute set in the system and N be the set $\{1, 2, \dots, n + 1\}$. The algorithm further randomly chooses $t_1, t_2, \dots, t_{n+1}$ from group $\mathbb{G}_1$ and defines a function T as:

$$T(X) = g^{X^n} \prod_{i=1}^{n+1} t_i^{\Delta_{i,N}(X)} \quad (4)$$

As in [4], the function T is viewed as the function $g^{X^n} g^{h(X)}$ for the polynomial h with degree n . Finally, the outputs of Setup algorithm are denoted as:

$$\begin{cases} PK = \mathbb{G}_1, \mathbb{G}_2, e, g_1, g, e(g_1, g), (t_1, \dots, t_{n+1}), H, N, T, \Delta_{i,S}(x) \\ MK = \alpha \end{cases} \quad (5)$$

4.2.2. Keygen

The Keygen algorithm outputs a private key for a data user u , who can use the private key to generate a legal search token with respect to some query keyword of interested. Before invoking Keygen algorithm, the data owner defines an access tree $\mathbb{T}_u$ with the root node r for u according to u 's system role. Next, when inputting the public parameter PK , the master key MK , and $\mathbb{A}_u$, the algorithm proceeds as follows. For each non-leaf node x in $\mathbb{T}_u$ (in a top-down manner), the algorithm first chooses a polynomial q_x and sets its degree d_x to be $d_x = k_x - 1$, where k_x is the node x 's threshold value in $\mathbb{T}_u$. For the root node r , set $q_r(0) = MK = y$ and choose d_r random values for other points of q_r to completely establish it. For any other non-leaf node x , set $q_x(0) = q_{parent(x)}(index(x))$ and choose d_x random values for other points of q_x to completely establish it. For each leaf node x in $\mathbb{T}_u$, choose a random value r_x and compute $G_1 = g^{r_x}$, $G_2 = g^{q_x(0)}$, and $\hat{T} = T(attr(x))^{r_x}$. Let L be the set of leaf nodes of the access tree $\mathbb{T}_u$. Finally, the outputs of Keygen algorithm are denoted as:

$$\mathbb{K}_u = (\forall x \in L : G_1 = g^{r_x}, G_2 = g^{q_x(0)}, \hat{T} = T(attr(x))^{r_x}), \quad (6)$$

which is transmitted to the data user u via secure channels.

4.2.3. Enc

The Enc algorithm outputs the ciphertext of an index keyword w . Before running the algorithm, the data owner specifies a set of attributes $\mathbb{A}$ for the index keyword w according to w 's access permission in the system. On inputting the public system PK , the index keyword w , and the corresponding attribute set $\mathbb{A}$, the algorithm proceeds as follows. It first chooses a random value $s \in \mathbb{Z}_p^*$ and computes

$\mathcal{I}' = e(g_1, g)^{sH(w)}$ and $\mathcal{I}'' = g^s$. Then, for each attribute $a \in \mathbb{A}$, the algorithm further computes $\mathcal{I}_a = T(a)^s$. Finally, the outputs of the Enc algorithm are denoted as:

$$\mathbb{I}(w) = (\mathbb{A}, \mathcal{I}' = e(g_1, g)^{sH(w)}, \mathcal{I}'' = g^s, \forall a \in \mathbb{A} : \mathcal{I}_a = T(a)^s) \quad (7)$$

4.2.4. Searchtoken

The Searchtoken algorithm outputs the search token $\mathbb{ST}_u(\hat{w})$ about a query keyword $\hat{w}$ that the data user u is interested in. On inputting the public system PK, the data user u 's key $\mathbb{K}_u$, and the search keyword $\hat{w}$, the algorithm proceeds as follows. For each key component in $\mathbb{K}_u$, it chooses a random value $\lambda \in \mathbb{Z}_p^*$ and encrypts the search keyword $\hat{w}$ to be $G_1^{H(\hat{w})\lambda}$ and $G_2^{H(\hat{w})} \cdot \hat{T}^{H(\hat{w})\lambda}$. Finally, the outputs of the Searchtoken algorithm are denoted as:

$$\begin{aligned} \mathbb{ST}_u(\hat{w}) &= (\forall x \in L, \mathcal{ST}_1 = G_1^{H(\hat{w})\lambda_x} = g^{r_x H(\hat{w})\lambda_x}, \\ \mathcal{ST}_2 &= G_2^{H(\hat{w})} \cdot \hat{T}^{H(\hat{w})\lambda_x} = g^{q_x(0)H(\hat{w})} \cdot T(attr(x))^{r_x H(\hat{w})\lambda_x}) \end{aligned} \quad (8)$$

4.2.5. Search

Given an encrypted index keyword $\mathbb{I}(w)$ and an encrypted search keyword $\mathbb{ST}_u(\hat{w})$ submitted by the data user u , the server invokes the Search algorithm to perform matching between $\mathbb{I}(w)$ and $\mathbb{ST}_u(\hat{w})$. The algorithm outputs 1 if and only if the following two conditions hold simultaneously: (1) the data user u has access to the index keyword w ; and (2) the search keyword $\hat{w}$ is identical to w . In the whole search process, the server knows nothing about the index keyword and the search keyword. Now, we give the concrete implementation for Search algorithm as follows. We first define a recursive algorithm $\text{SearchNode}(\mathbb{I}(w), \mathbb{ST}_u(\hat{w}), x)$, where x denotes a node in the access tree (in our scheme, the access tree is embedded in u 's private key encrypting the search keyword $\hat{w}$, therefore it is actually ultimately implanted in the corresponding search token). It outputs an element of $\mathbb{G}_2$ or $\perp$ as follows. For each leaf node x in the access tree, and letting $a = attr(x)$, if $a \in \mathbb{A}$, then:

$$\begin{aligned} \text{SearchNode}(\mathbb{I}(w), \mathbb{ST}_u(\hat{w}), x) &= \frac{e(\mathcal{ST}_2, \mathcal{I}'')}{e(\mathcal{ST}_1, \mathcal{I}_a)} = \frac{e(G_2^{H(\hat{w})} \cdot \hat{T}^{H(\hat{w})\lambda_x}, g^s)}{e(G_1^{H(\hat{w})\lambda_x}, T(a)^s)} \\ &= \frac{e(g^{q_x(0)H(\hat{w})} \cdot T(a)^{r_x H(\hat{w})\lambda_x}, g^s)}{e(g^{r_x H(\hat{w})\lambda_x}, T(a)^s)} \\ &= \frac{e(g^{q_x(0)H(\hat{w})}, g^s) \cdot e(T(a), g)^{r_x H(\hat{w})\lambda_x s}}{e(T(a), g)^{r_x H(\hat{w})\lambda_x s}} \\ &= e(g, g)^{sq_x(0)H(\hat{w})} \end{aligned} \quad (9)$$

if $a = attr(x) \notin \mathbb{A}$, we define $\text{SearchNode}(\mathbb{I}(w), \mathbb{ST}_u(\hat{w}), x) = \perp$. On the other hand, for each non-leaf node x , the algorithm $\text{SearchNode}(\mathbb{I}(w), \mathbb{ST}_u(\hat{w}), x)$ then proceeds as follows: for all nodes z that are children of x , it runs $\text{SearchNode}(\mathbb{I}(w), \mathbb{ST}_u(\hat{w}), z)$ and stores the outputs as F_z . Let S_x be an arbitrary k_x -sized set of child nodes z such that $F_z \neq \perp$. If no such set exists, then the node is not satisfied and the function returns $\perp$ and Search algorithm returns 0, which means the data user u has not the search permission of the index keyword w .

Otherwise, we compute:

$$\begin{aligned}
F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}, \text{ where } i = \text{index}(z), S'_x = (\forall z \in S_x : \text{index}(z)) \\
&= \prod_{z \in S_x} (e(g, g)^{s \cdot q_z(0)H(\hat{w})})^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} (e(g, g)^{s \cdot q_{\text{parent}(z)}(\text{index}(z)) \cdot H(\hat{w})})^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} e(g, g)^{s \cdot q_x(i) \cdot H(\hat{w}) \cdot \Delta_{i,S'_x}(0)} \\
&= e(g, g)^{sq_x(0)H(\hat{w})} \quad (\text{using polynomial interpolation})
\end{aligned} \tag{10}$$

Therefore, through the above recursive computation, we can observe that, if the attribute set $\mathbb{A}$ in $\mathbb{I}_u$ satisfies the access tree embedded in the search token, then, for the root node r , $\text{SearchNode}(\mathbb{I}(w), \mathbb{ST}_u(\hat{w}), r) = e(g, g)^{sq_r(0)H(\hat{w})}$, which means the data user u is allowed to access the index keyword w . Finally, the matching between the index keyword w and the search keyword $\hat{w}$ is performed through the following comparison:

$$\text{SearchNode}(\mathbb{I}(w), \mathbb{ST}_u(\hat{w}), r) = \mathcal{I}' \tag{11}$$

If Equation (10) holds, then $w = \hat{w}$ and the Search algorithm returns 1; otherwise, $w \neq \hat{w}$ and the Search algorithm returns 0. We can verify the matching correctness by the simple derivation as follows:

$$\begin{aligned}
\text{SearchNode}(\mathbb{I}(w), \mathbb{ST}_u(\hat{w}), r) &= e(g, g)^{sq_r(0)H(\hat{w})} = e(g, g)^{syH(\hat{w})} \\
&= e(g^y, g)^{sH(\hat{w})} = e(g_1, g)^{sH(\hat{w})} \\
&= \mathcal{I}'
\end{aligned} \tag{12}$$

5. Performance Analysis

In this section, we analyze the performance for our scheme from two aspects: storage cost and computation cost. Meanwhile, we also provide a performance comparison between the similar work KP-ABKS proposed in [49] and ours. For ease of description, we first define some notations, as shown in Table 1, where we are only interested in those relatively time-consuming operations such as pairing operation and exponentiation operation, while ignoring extremely efficient operations such as multiplication operations [51].

Table 1. Notations used in complexity analysis.

Notations	Description
P	Time cost of a pairing computation.
$E_{\mathbb{G}_1}$	Time cost of an exponentiation in group $\mathbb{G}_1$.
$E_{\mathbb{G}_2}$	Time cost of an exponentiation in group $\mathbb{G}_2$.
$C_{\mathbb{G}_1}$	Time cost of choosing a random element from $\mathbb{G}_1$.
H	Time cost of hashing a bit-string to an element of $\mathbb{G}_1$.
$ x $	The cardinality if x is a set; the bit-length if x is an element.
L	Leaf-node set of the access tree in the data user's private key.
$\mathbb{A}$	Attribute set of encrypting an index keyword.
S	Least interior node set satisfying an access structure (include the root).
U	Universe set of attributes.

The time cost that each algorithm spends in our scheme and in KP-ABKS [49] is shown in Table 2. In Table 2, we can observe that KPSABES and KP-ABKS have the same computation complexity with respect to Keygen algorithm. More importantly, our scheme outperforms KP-ABKS on Enc algorithm

and Search algorithm due to the less pairing operations and exponentiation operations compared with KP-ABKS.

Table 2. The computation cost of algorithms in KPSABES and KP-ABKS [49].

Algorithm	KPSABES	KP-ABKS [49]
Setup	$(U + 2)C_{G_1} + E_{G_1} + P$	$(U + 3)C_{G_1} + 3E_{G_1}$
Keygen	$ L (H + 3E_{G_1})$	$ L (H + 3E_{G_1})$
Enc	$E_{G_2} + \mathbb{A} H + (\mathbb{A} + 1)E_{G_1}$	$ \mathbb{A} H + (\mathbb{A} + 4)E_{G_1}$
Searchtoken	$3 L E_{G_1}$	$(2 L + 3)E_{G_1}$
Search	$2 S P + S E_{G_2}$	$(2 S + 4)P + S E_{G_2}$

We use the output size of the algorithm to evaluate the storage cost for KPSABES and KP-ABKS, as shown in Table 3. We can see that, compared with KP-ABKS, in addition to Keygen algorithm, other algorithms in KPSABES need fewer spaces to store their outcomes. Here, we do not consider the storage cost of search results, thus the output size of Search algorithm is ignored.

Table 3. The storage cost of the output of each algorithm in KPSABES and KP-ABKS [49].

Algorithm	KPSABES	KP-ABKS [49]
Setup	$(U + 2) G_1 + G_2 + Z_p^* $	$(U + 5) G_1 + 3 Z_p^* $
Keygen	$3 L G_1 $	$2 L G_1 $
Enc	$ G_2 + (\mathbb{A} + 1) G_1 $	$(\mathbb{A} + 3) G_1 $
Searchtoken	$2 L G_1 $	$(2 L + 2)E_{G_1}$
Search	/	/

6. Security Proof

In this section, we provide formal security proof for our proposed KPSABES based on the selectively chosen-keyword attack model and the adaptively chosen-plaintext attack model.

Theorem 1. *If DBDH is a complex problem, the index keyword encryption algorithm Enc in KPSABES is selectively secure against chosen-keyword attack.*

Proof. If there is a polynomial-time adversary $\mathcal{A}$ that can break KPSABES with a non-negligible advantage ϵ to gain the index keyword from the corresponding ciphertext, a simulator $\mathcal{B}$ can be constructed to solve DBDH problem with advantage $\frac{\epsilon}{2}$.

The challenger $\mathcal{C}$ first runs Setup algorithm to set up the system public parameter PK and the master key MK. Then, $\mathcal{C}$ flips a fair binary coin $\nu \in \{0, 1\}$ and transmits a tuple t_ν to the simulator $\mathcal{B}$. If $\nu = 0$, the tuple t_0 is set to be $(g, A = g^a, B = g^b, C = g^c, Z = e(g, g)^{abc})$; on the contrary, tuple t_1 is set as $(g, A = g^a, B = g^b, C = g^c, Z = e(g, g)^z)$, where a, b, c, z are randomly chosen from $\mathbb{Z}_p^*$. Next, the challenger delegates the simulator $\mathcal{B}$ to play the, h1Selectively Chosen-Keyword Attack Game with $\mathcal{A}$ as follows.

Setup. The adversary $\mathcal{A}$ specifies a set of attributes $\mathbb{A}^*$ with n elements that he wishes to be challenged upon and sends it to the simulator $\mathcal{B}$. $\mathcal{B}$ randomly chooses an n degree polynomial $f(X)$ and constructs the other n degree polynomial $u(X)$ as follows: for all $X \in \mathbb{A}^*$ set $u(X) = -X^n$ and otherwise set $u(X) \neq -X^n$, which can ensure that if and only if $X \in \mathbb{A}^*$, $u(X) = -X^n$ holds for all X . Next, $\mathcal{B}$ sets $t_i = g^{u(i)}g^{f(i)}$ for $\forall i \in [1, n + 1]$. Since $f(X)$ is randomly chosen, the value t_i is also random. Finally, we have $T(i) = g^{i^n + u(i)}g^{f(i)}$.

Phase 1. The adversary $\mathcal{A}$ issues private key queries for many times corresponding to access trees $T_1, \dots, T_q$ such that none of them are satisfied by the challenge attribute set $\mathbb{A}^*$. For any a requested access tree $T^* \in [T_1, T_q]$, since the challenge attribute set $\mathbb{A}^*$ does not satisfy T^* , i.e., $T^*(\mathbb{A}^*) = 0$,

to generate the corresponding private key, $\mathcal{B}$ has to assign a polynomial Q_x of degree d_x for every node x in the access tree T^* . To do that, we first define two procedures PolySat and PolyUnsat as follows.

- PolySat($T_x^*, \mathbb{A}^*, \lambda_x$). PolySat takes as input an access tree T_x^* rooted at x such that $T_x^*(\mathbb{A}^*) = 1$, the attribute set $\mathbb{A}^*$, a secret value $\lambda_x \in \mathbb{Z}_p^*$, and determines the polynomials for each node of T_x^* . Specifically, it first generates a polynomial q_x of degree d_x for the root node x such that $q_x(0) = \lambda_x$ and randomly sets d_x other points to completely generate q_x . Then, for each child node x' of x , it calls PolySat($T_{x'}^*, \mathbb{A}^*, q_x(\text{index}(x'))$), where $q_{x'}(0) = q_x(\text{index}(x'))$.
- PolyUnsat($T_x^*, \mathbb{A}^*, g^{\lambda_x}$). PolyUnsat takes as input an access tree T_x^* rooted at x such that $T_x^*(\mathbb{A}^*) = 0$, the attribute set $\mathbb{A}^*$, a group element $g^{\lambda_x} \in \mathbb{G}_1$, $\lambda_x \in \mathbb{Z}_p^*$, and determines the polynomials for each node of T_x^* . Specifically, it first generates a polynomial q_x of degree d_x for the root node x and makes $g^{q_x(0)} = g^{\lambda_x}$. $T_x^*(\mathbb{A}^*) = 0$ means that no more than d_x children of node x are satisfied. Let SC denote a set of child nodes of x that are satisfied by $\mathbb{A}^*$, we have $|\text{SC}| < d_x$. For each node $x' \in \text{SC}$, it randomly chooses a value $\lambda_{x'} \in \mathbb{Z}_p^*$ and sets $q_x(\text{index}(x')) = \lambda_{x'}$ and randomly chooses $d_x - |\text{SC}|$ other points to completely define q_x . Then, the procedure continues to run PolySat($T_{x'}^*, \mathbb{A}^*, q_x(\text{index}(x'))$) if $x' \in \text{SC}$, where $q_x(\text{index}(x'))$ is known to the simulator; otherwise, the PolyUnsat($T_{x'}^*, \mathbb{A}^*, g^{q_x(\text{index}(x'))}$) is run, where $g^{q_x(\text{index}(x'))}$ is known to the simulator.

Now, the simulator $\mathcal{B}$ runs PolyUnsat($T^*, \mathbb{A}^*, A$), where $A = g^a$. After the procedure call finishes, $\mathcal{B}$ can determine two kinds of results about the polynomial: (1) for the root node r of T^* , $q_r(0) = a$; and (2) for each leaf node x of T^* , if $\text{attr}(x) \in \mathbb{A}^*$, $\mathcal{B}$ gets the complete polynomial q_x and can further compute $q_x(0)$; if $\text{attr}(x) \notin \mathbb{A}^*$, $\mathcal{B}$ gets $g^{q_x(0)}$.

Finally, $\mathcal{B}$ generates the private key corresponding to the access tree T^* based on the polynomial of each leaf node x as follows. Let $\text{attr}(x) = i$.

- If $i \in \mathbb{A}^*$, $\mathcal{B}$ chooses a random value $r_x \in \mathbb{Z}_p^*$ and sets $G_1 = g^{r_x}$, $G_2 = g^{q_x(0)}$, and $\hat{T} = T(\text{attr}(x))^{r_x} = T(i)^{r_x}$.
- If $i \notin \mathbb{A}^*$, let $g_2 = g^{q_x(0)}$, $\mathcal{B}$ chooses a random value $r'_x \in \mathbb{Z}_p^*$ and sets the key components corresponding to the node x as follows.

$$\begin{cases} G_1 = g_2^{\frac{-1}{i^n+u(i)}} g^{r'_x} \\ G_2 = g_2 \\ \hat{T} = g_2^{\frac{-f(i)}{i^n+u(i)}} (g^{i^n+u(i)} g^{f(i)})^{r'_x} / G_2 \end{cases}$$

If we let $r_x = r'_x - \frac{q_x(0)}{i^n+u(i)}$, the above keys are legitimate, because

$$\begin{aligned} G_1 &= g_2^{\frac{-1}{i^n+u(i)}} g^{r'_x} = g^{\frac{-q_x(0)}{i^n+u(i)}} g^{r'_x} = g^{r'_x - \frac{q_x(0)}{i^n+u(i)}} = g^{r_x} \\ G_2 &= g_2 = g^{q_x(0)} \\ \hat{T} &= g_2^{\frac{-f(i)}{i^n+u(i)}} (g^{i^n+u(i)} g^{f(i)})^{r'_x} / G_2 = g^{\frac{-q_x(0)}{i^n+u(i)}} (g^{i^n+u(i)} g^{f(i)})^{r'_x} / G_2 \\ &= g^{q_x(0)} (g^{i^n+u(i)} g^{f(i)})^{\frac{r'_x}{i^n+u(i)}} (g^{i^n+u(i)} g^{f(i)})^{r'_x} / G_2 \\ &= g^{q_x(0)} (g^{i^n+u(i)} g^{f(i)})^{r'_x - \frac{q_x(0)}{i^n+u(i)}} / g^{q_x(0)} \\ &= (g^{i^n+u(i)} g^{f(i)})^{r'_x - \frac{q_x(0)}{i^n+u(i)}} \\ &= T(i)^{r_x} \end{aligned}$$

Therefore, the simulator can generate the correct private key for the requested access tree T^* , which is sent to the adversary $\mathcal{A}$. The adversary can use the private key to encrypt any query keyword $\hat{w}$ to generate valid search token.

Challenge. The adversary $\mathcal{A}$ outputs two keywords w_0 and w_1 and sends them to the simulator. $\mathcal{B}$ randomly chooses a bit $b \in \{0, 1\}$ and encrypts w_b with $\mathbb{A}^*$ as:

$$\mathbb{I}^*(w_b) = (\mathbb{A}^*, \mathcal{I}' = Z^{H(w_b)}, \mathcal{I}'' = g^s, \forall a \in \mathbb{A}^* : \mathcal{I}_a = T(a)^s),$$

where s is chosen at random from $\mathbb{Z}_p^*$. If $\nu = 0$, tuple $t_0 : (g, A = g^a, B = g^b, C = g^c, Z = e(g, g)^{abc})$ is sent to $\mathcal{B}$, w_b 's ciphertext can be denoted as $\mathbb{I}(w_b) = (\mathbb{A}^*, \mathcal{I}' = e(g, g)^{abcH(w_b)}, \mathcal{I}'' = g^s, \forall a \in \mathbb{A}^* : \mathcal{I}_a = T(a)^s)$. Since a, b, c, s are all random from the adversary $\mathcal{A}$'s view, we let $bc = s$. Thus, the w_b 's ciphertext is $\mathbb{I}^*(w_b) = (\mathbb{A}^*, \mathcal{I}' = e(g, g)^{asH(w_b)}, \mathcal{I}'' = g^s, \forall a \in \mathbb{A}^* : \mathcal{I}_a = T(a)^s)$, which can further be denoted as $\mathbb{I}^*(w_b) = (\mathbb{A}^*, \mathcal{I}' = e(g, g)^{sH(w_b)}, \mathcal{I}'' = g^s, \forall a \in \mathbb{A}^* : \mathcal{I}_a = T(a)^s)$. This is a valid ciphertext for w_b under $\mathbb{A}^*$. On the other hand, if $\nu = 1$, tuple $t_1 : (g, A = g^a, B = g^b, C = g^c, Z = e(g, g)^z)$ is sent to $\mathcal{B}$. The ciphertext of w_b can be denoted as $\mathbb{I}^*(w_b) = (\mathbb{A}^*, \mathcal{I}' = e(g, g)^{zH(w_b)}, \mathcal{I}'' = g^s, \forall a \in \mathbb{A}^* : \mathcal{I}_a = T(a)^s)$. Obviously, $\mathcal{I}'$ is a random element of $\mathbb{G}_2$ and contains no information about w_b .

Phase 2. The adversary $\mathcal{A}$ continues to issue the private key queries m times corresponding to access trees $T_{q+1}, \dots, T_{q+m}$, as in Phase 1.

Guess. The adversary $\mathcal{A}$ outputs the guess b' of b . If $b = b'$, then $\mathcal{B}$ outputs $\nu' = 0$, it means that $\mathcal{B}$ obtains the valid DBDH tuple from $\mathcal{C}$. Since $\mathcal{A}$ has advantage ϵ to break KPSABES and recover the index keyword information from the ciphertext, the probability that $\mathcal{A}$ outputs $b' = b$ is $\frac{1}{2} + \epsilon$. If $b \neq b'$, then $\mathcal{B}$ outputs $\nu' = 1$ and the challenger $\mathcal{C}$ sends the random tuple to $\mathcal{B}$. The probability that $\mathcal{A}$ outputs $b' = b$ is $\frac{1}{2}$.

The overall advantage that $\mathcal{B}$ solves the DBDH problem is as follows:

$$\begin{aligned} & \left| \frac{1}{2} \Pr[\nu = \nu' | \nu = 0] + \frac{1}{2} \Pr[\nu = \nu' | \nu = 1] - \frac{1}{2} \right| \\ &= \left| \left[\frac{1}{2} \left(\frac{1}{2} + \epsilon \right) + \frac{1}{2} \cdot \frac{1}{2} \right] - \frac{1}{2} \right| = \frac{\epsilon}{2} \end{aligned} \quad (13)$$

Therefore, $\mathcal{B}$ can solve the DBDH problem with the non-negligible advantage $\frac{\epsilon}{2}$, which contradicts the recognized complexity assumption of DBDH problem. $\square$

Theorem 2. If DDH is a complex problem, the search keyword encryption algorithm Searchtoken in KPSABES is semantically secure against adaptively chosen-plaintext attack.

Proof. If there is a polynomial-time adversary $\mathcal{A}$ that can break KPSABES with a non-negligible advantage ϵ to recover the search keyword information from the corresponding ciphertext, a simulator $\mathcal{B}$ can be constructed to solve DDH problem with advantage $\frac{\epsilon}{2}$.

$\mathcal{C}$ flips a fair binary coin $\nu \in \{0, 1\}$ and transmits a tuple t_ν to the simulator $\mathcal{B}$. If $\nu = 0$, the tuple t_0 is set to be $(g, A = g^a, B = g^b, C = g^{ab})$; on the contrary, tuple t_1 is set as $(g, A = g^a, B = g^b, C = g^c)$, where a, b, c are random elements in $\mathbb{Z}_p^*$. Next, the challenger delegates the simulator $\mathcal{B}$ to play the adaptively chosen-plaintext attack game with the adversary $\mathcal{A}$ as follows.

Setup. The simulator $\mathcal{B}$ generates a public parameter PK and a master key MK. Then, $\mathcal{B}$ sends PK to $\mathcal{A}$.

Phase 1. The adversary $\mathcal{A}$ adaptively queries for the ciphertext of any search keyword.

Challenge. The adversary $\mathcal{A}$ outputs two keywords w_0 and w_1 and sends them to $\mathcal{B}$. $\mathcal{B}$ randomly chooses a bit $b \in \{0, 1\}$ and encrypts w_b under an access tree $\mathbb{A}$ with leaf node set L as:

$$\mathbb{ST}_{\mathcal{B}}^*(w_b) = (\mathcal{ST}_1 = C^{H(w_b)}, \mathcal{ST}_2 = A^{H(w_b)} \cdot C^{H(w_b)})$$

which is sent to $\mathcal{A}$. If tuple $t_0 : (g, A = g^a, B = g^b, C = g^{ab})$ is sent to $\mathcal{B}$, w_b 's ciphertext can be denoted as $\mathbb{ST}_{\mathcal{B}}^*(w_b) = (\mathcal{ST}_1 = g^{abH(w_b)}, \mathcal{ST}_2 = g^{aH(w_b)} \cdot g^{abH(w_b)})$ (Note that since the ciphertext structures of all leaf nodes are identical, we uniformly uses the generator g of $\mathbb{G}_1$ to present function value $T(attr(x)) \in \mathbb{G}_1$ for any leaf node $x \in L$). For example, given a leaf node $x \in L$, if we let $a = r_x = q_0(x)$, $b = \lambda_x$, then this is a valid ciphertext component. On the other hand, if tuple $t_1 : (g, A = g^a, B = g^b, C = g^c)$ is sent to $\mathcal{B}$, w_b 's ciphertext is $\mathbb{ST}_{\mathcal{B}}^*(w_b) = (\mathcal{ST}_1 = g^{cH(w_b)}, \mathcal{ST}_2 = g^{aH(w_b)} \cdot g^{cH(w_b)})$. In this case, $\mathcal{ST}_1 = g^{cH(w_b)}$ and $\mathcal{ST}_2 = g^{aH(w_b)} \cdot g^{cH(w_b)}$ are two random elements of $\mathbb{G}_1$ from the adversary $\mathcal{A}$'s view and contain no information about w_b .

Phase 2. The adversary $\mathcal{A}$ continues to ask the simulator $\mathcal{B}$ for the ciphertext of any search keyword, including w_0 and w_1 .

Guess. The adversary $\mathcal{A}$ outputs the guess b' of b . If $b = b'$, then $\mathcal{B}$ outputs $v' = 0$, it means that $\mathcal{B}$ obtains the valid DDH tuple from $\mathcal{C}$. Since $\mathcal{A}$ has advantage ϵ to recover the search keyword information from the search token, the probability that $\mathcal{A}$ outputs $b' = b$ is $\frac{1}{2} + \epsilon$. If $b \neq b'$, then $\mathcal{B}$ outputs $v' = 1$ and the challenger $\mathcal{C}$ sends the random tuple to $\mathcal{B}$. The probability that $\mathcal{A}$ outputs $b' = b$ is $\frac{1}{2}$.

The overall advantage that $\mathcal{B}$ solves the DDH problem is as follows:

$$\begin{aligned} & \left| \frac{1}{2}Pr[v = v' | v = 0] + \frac{1}{2}Pr[v = v' | v = 1] - \frac{1}{2} \right| \\ &= \left| \left[\frac{1}{2} \left(\frac{1}{2} + \epsilon \right) + \frac{1}{2} \cdot \frac{1}{2} \right] - \frac{1}{2} \right| = \frac{\epsilon}{2} \end{aligned} \quad (14)$$

Therefore, $\mathcal{B}$ can solve the DDH problem with a non-negligible advantage $\frac{\epsilon}{2}$, which contradicts the recognized complexity assumption of DDH problem. $\square$

7. Experimental Evaluation

We evaluated the performances for our KPSABES and the similar work KP-ABKS proposed in [49] through extensive experiments on a real dataset, the Request For Comments Database (RFC) [52]. We chose 2000 data files from RFC and extracted 320 index keywords from these data files by using Hermetic Word Frequency Counter [53] tool. All programs were run in the Java platform with the help of JPBC library [51]. Specifically, the evaluations of Setup algorithm, Keygen algorithm, Enc algorithm, and Searchtoken algorithm were conducted in a client environment: a Windows 7 desktop system with 2.3-GHz Intel Core (TM) i5-6200U and 4-GB RAM. The server environment was used to evaluate the performance of Search algorithm: an Ubuntu 16.04 system with 3.60-GHz Intel Core (TM) i7-7700 CPU and 8-GB RAM.

7.1. Evaluation of Setup Algorithm

Figure 3 shows the time cost of Setup algorithm in KPSABES and KP-ABKS. Randomly generating a group element of $\mathbb{G}_1$ was relatively time-consuming operation from the bilinear elliptic curve group, needing about 200 millisecond in our test environment. We can see that the time cost of both schemes increased linearly with the number of system attributes. However, our scheme had a slight advantage to set up the system running environment. An important reason was that the Setup algorithm in KP-ABKS needed more exponentiation operations compared with our KPSABES.

Figure 3. The time cost of Setup algorithm in KPSABES and KP-ABKS.

7.2. Evaluation of Keygen Algorithm

Figure 4 shows the time cost of Keygen algorithm in KPSABES and KP-ABKS. The most expensive operation in this algorithm was the hash operation that hashing a leaf node to an element of $\mathbb{G}_1$ and needed about 106 millisecond at average. As shown in Figure 4, these two schemes had approximatively identical time cost on Keygen algorithm. Further, they were linear to the number of leaf nodes in the access tree. The experimental results are also in accordance with our algorithm complexity analysis in Section 5.

Figure 4. The time cost of Keygen algorithm in KPSABES and KP-ABKS.

7.3. Evaluation of Enc Algorithm

Figure 5 shows the time cost of Enc algorithm in KPSABES and KP-ABKS. To thoroughly evaluate the performance of the index keyword encryption, we made three groups of different tests in our experimental dataset. Figure 5a demonstrates the time cost on one index keyword encryption when varying the number of attributes. We can see that the time cost of both schemes increased linearly with the number of attributes, which is reasonable since they are both inherited from the original KP-ABE scheme in [4]. Figure 5b shows the time cost of encrypting index keywords with different size when fixing the number of attributes and data files. The experimental result shows that encrypting all 320 index keywords extracted from 2000 data files consumed about 290 s and 408 s in KPSABES and KP-ABKS, respectively, when the number of attributes was set to be 6. Figure 5c demonstrates that the number of data files had no influence on index keyword encryption in the two schemes when fixing the size of index keywords and the number of attributes. As shown in Figure 5, t our KPSABES

was more efficient on index keyword encryption compared with KP-ABKS. Moreover, the greater was the number of index keywords, the more obvious was the advantage. This results is reasonable because encrypting one index keyword in KP-ABKS needed two more exponential operations than our scheme according to the algorithm complexity analysis. In addition, as shown in Figure 5b,c, the index keyword encryptions in the whole dataset were very time-consuming, but was a one-time operation.

Figure 5. Time cost of Enc algorithm in KPSABES and KP-ABKS: (a) time cost of encrypting one index keyword for different number of attributes; (b) time cost of encrypting index keywords for different number of index keywords with fixed number of attributes $|\mathbb{A}| = 6$ and fixed number of data files, $n = 2000$; and (c) time cost of encrypting index keywords for different number of data files with fixed number of attributes $|\mathbb{A}| = 6$ and fixed number of index keyword, $i = 200$.

7.4. Evaluation of Searchtoken Algorithm

Figure 6 shows the time cost of Searchtoken algorithm in KPSABES and KP-ABKS. We can see that, in KPSABES and KP-ABKS, the time spent on the search keyword encryption was closely related to the number of leaf nodes in the access tree embedded in the data user's private key. In addition, when the number of leaf nodes in the access tree was more than 3, our scheme needed more time to generate a search token. This was because KP-ABKS and KPSABES needed $2l + 3$ and $3l$ exponentiation operations to encrypt a search keyword, respectively, where l denotes the number of leaf nodes in the access tree. However, these two schemes were extremely efficient on search keyword encryption. For example, when the number of leaf nodes in the access tree was 10, the time cost of the search keyword encryption was about 1.356 s in KPSABES and 0.968 s in KP-ABKS, which is acceptable in practical application.

Figure 6. The time cost of Searchtoken algorithm in KPSABES and KP-ABKS.

7.5. Evaluation of Search Algorithm

Figure 7 shows the time cost of Search algorithm in KPSABES and KP-ABKS. Since the search performance is the key index in a search system, we made six groups of different tests with different experimental parameters to evaluate the search performance for KPSABES and KP-ABKS. Specifically, Figure 7a–c demonstrates the average time cost for different number of index keywords with fixed size of data files ($n = 2000$) when setting $|S| = 2$, $|S| = 4$, and $|S| = 6$, respectively, where S denotes the least interior node set satisfying the access tree (include the root). Figure 7d–f shows the average time cost for different size of data files with fixed number of index keywords ($i = 200$) when setting $|S| = 2$, $|S| = 4$, and $|S| = 6$, respectively. We observed that the time cost of both schemes was determined by not only the number of index keywords but also the number of least interior nodes satisfying the access tree and was insensitive to the size of data files, which confirmed our complexity analysis in Section 5. Moreover, as shown in Figure 7, our scheme outperformed KP-ABKS on search performance in an obvious advantage due to requiring fewer pairing computations.

Figure 7. Time cost of Search algorithm in KPSABES and KP-ABKS.

8. Conclusions

In this paper, based on the original KP-ABE scheme [4], we design a key-policy searchable attribute-based encryption scheme (KPSABES) to support efficient keyword search and fine-grained access control over encrypted data. KPSABES is very suitable for the cryptography based data sharing storage system that needs the data access control and keyword based data searching. Unlike the similar KP-ABKS scheme proposed in [49], on the basis of the scheme in [4], our scheme does not require introducing any extra ciphertext components and expensive operations to support data searching. Therefore, KPSABES has some obvious advantages in terms of storage and computation cost compared with KP-ABKS. In addition, extensive experiments on a real dataset demonstrated that KPSABES is superior in many aspects to KP-ABKS, especially in the search performance. As our future work, we will consider the problem of efficient multi-keyword ranked search with fine-grained access control over encrypted data.

Author Contributions: Conceptualization, Y.X.; methodology, H.Y. and J.Z.; software, J.Z. and L.O.; validation, H.Y.; formal analysis, H.Y. and S.L.; investigation, H.Y. and J.Z.; resources, Z.Q. and S.L.; data curation, L.O. and J.Z.; writing—original draft preparation, H.Y. and Y.X.; writing—review and editing, L.O.; visualization, H.Y.; supervision, Z.Q. and S.L.; and project administration, Z.Q. and S.L.

Funding: This research was funded by the Natural Science Foundation of Hunan Province under Grant No. 2017JJ2292; Science and Technology Key Projects of Hunan Province under Grant No. 2016JC2012; Outstanding Youth Research Project of Provincial Education Department of Hunan under Grant No. 17B030; and Science and Technology Planning Project of Changsha under Grant No. K1705018, ZD1601042 and K1705031.

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

References

- Ren, K.; Wang, C.; Wang, Q. Security Challenges for the Public Cloud. *IEEE Internet Comput.* **2012**, *16*, 69–73. [\[CrossRef\]](#)
- Kamara, S.; Lauter, K. Cryptographic cloud storage. In *Financial Cryptography and Data Security*; Springer: Berlin/Heidelberg, Germany, 2010.
- Peng, Y.; Zhao, W.; Xie, F.; Dai, Z.; Gao, Y.; Chen, D. Secure cloud storage based on cryptographic techniques. *J. China Univ. Posts Telecommun.* **2012**, *19*, 182–189. [\[CrossRef\]](#)
- Goyal, V.; Pandey, O.; Sahai, A.; Waters, B. Attribute-based encryption for fine-grained access control of encryption data. In Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, 30 October–3 November 2006; pp. 89–98.
- Song, D.; Wagner, D.; Perrig, A. Practical techniques for searches on encrypted data. In Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 14–17 May 2000; Volume 8, pp. 44–55.
- Curtmola, R.; Garay, J.; Kamara, S.; Ostrovsky, R. Searchable symmetric encryption: Improved definitions and efficient constructions. In Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 30 October–3 November 2006; Volume 19, pp. 79–88.
- Kamara, S.; Papamanthou, C.; Roeder, T. Dynamic searchable symmetric encryption. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012; pp. 965–976.
- Stefanov, E.; Papamanthou, C.; Shi, E. Practical Dynamic Searchable Encryption with Small Leakage. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2014.
- Kim, K.S.; Kim, M.; Lee, D.; Park, J.H.; Kim, W.H. Forward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1449–1463.
- Bost, R.; Minaud, B.; Ohrimenko, O. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1465–1482.
- Kurosawa, K.; Ohtaki, Y. UC-Secure Searchable Symmetric Encryption. In *Financial Cryptography and Data Security*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 285–298.
- Yin, H.; Qin, Z.; Zhang, J.; Ou, L.; Li, K. Achieving secure, universal, and fine-grained query results verification for secure search scheme over encrypted cloud data. *IEEE Trans. Cloud Comput.* **2017**. [\[CrossRef\]](#)
- Zhang, Y.; Deng, R.H.; Shu, J.; Yang, K.; Zheng, D. TKSE: Trustworthy Keyword Search over Encrypted Data with Two-side Verifiability via Blockchain. *IEEE Access* **2018**, *6*, 31077–31087. [\[CrossRef\]](#)
- Hu, S.; Cai, C.; Wang, Q.; 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, 16–19 April 2018; pp. 792–800.
- Cao, N.; Wang, C.; Li, M.; Ren, K.; Lou, W. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 222–233. [\[CrossRef\]](#)
- Sun, W.; Wang, B.; Cao, N.; Li, M.; Lou, W.; Hou, Y.T. Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 3025–3035. [\[CrossRef\]](#)

17. Wang, B.; Yu, S.; Luo, W.; Hou, Y. Privacy-Preserving Multi-Keyword Fuzzy Search over Encrypted Data in the Cloud. In Proceedings of the IEEE INFOCOM—IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; pp. 2112–2120.
18. 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](#)]
19. Li, H.; Yang, Y.; Luan, T.; Liang, X.; Zhou, L.; Shen, X. Enabling Fine-grained Multi-keyword Search Supporting Classified Sub-dictionaries over Encrypted Cloud Data. *IEEE Trans. Dependable Secure Comput.* **2016**, *13*, 312–325. [[CrossRef](#)]
20. Fu, Z.; Ren, K.; Shu, J.; Sun, X.; Huang, F. Enabling Personalized Search over Encrypted Outsourced Data with Efficiency Improvement. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 2546–2559. [[CrossRef](#)]
21. Zhang, Q.; Liu, Q.; Shu, J.; Wang, G. PRMS: A Personalized Mobile Search over Encrypted Outsourced Data. *IEEE Access* **2018**, *6*, 31541–31552. [[CrossRef](#)]
22. Boneh, D.; Crescenzo, G.D.; Ostrovsky, R.; Persiano, G. Public-key encryption with keyword search. In *Advances in Cryptology—EUROCRYPT 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 506–522.
23. Golle, P.; Staddon, J.; Waters, B. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 31–45.
24. Ballard, L.; Kamara, S.; Monrose, F. Achieving efficient conjunctive keyword searches over encrypted data. In *Information and Communications Security*; Springer: Berlin/Heidelberg, Germany, 2005.
25. Boneh, D.; Waters, B. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 535–554.
26. Yin, H.; Qin, Z.; Zhang, J.; Li, W.; Ou, L.; Hu, Y.; Li, K. Secure Conjunctive Multi-keyword Search for Multiple Data Owners in Cloud Computing. In Proceedings of the 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan, China, 13–16 December 2016; pp. 761–768.
27. Sahai, A.; Waters, B. Fuzzy Identity-Based Encryption. In *Advances in Cryptology—EUROCRYPT 2005*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 457–473.
28. Bethencourt, J.; Sahai, A.; Waters, B. Ciphertext-policy attribute-based encryption. In Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP'07), Berkeley, CA, USA, 20–23 May 2007; pp. 321–334.
29. Cheung, L.; Newport, C. Provably Secure Ciphertext Policy ABE. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 29 October–2 November 2007; pp. 456–465.
30. Waters, B. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Public Key Cryptography*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 53–70.
31. Lewko, A.B.; Waters, B. Decentralizing attribute-based encryption. In *Advances in Cryptology—EUROCRYPT 2011*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 547–567.
32. Ostrovsky, R.; Sahai, A.; Waters, B. Attribute-based encryption with non-monotonic access structures. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 29 October–2 November 2007; pp. 195–203.
33. Chase, M. Multi-authority attribute based encryption. In *Theory of Cryptography*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 515–534.
34. Chase, M.; Chow, S.S.M. Improving privacy and security in multi-authority attribute-based encryption. In Proceedings of the 16th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 9–13 November 2009; pp. 121–130.
35. Kapadia, A.; Tsang, P.P.; Smith, S.W. Attribute-based publishing with hidden credentials and hidden policies. In Proceedings of the Network and Distributed System Security Symposium, NDSS 2007, San Diego, CA, USA, 28 February–2 March 2007; pp. 179–192.
36. Nishide, T.; Yoneyama, K.; Ohta, K. ABE with partially hidden encryptor-specified access structure. In *Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 111–129.
37. Zhang, Y.; Zheng, D.; Deng, R.H. Security and Privacy in Smart Health: Efficient Policy-Hiding Attribute-Based Access Control. *IEEE Internet Things J.* **2018**, *5*, 2130–2145. [[CrossRef](#)]
38. Zhang, Y.; Chen, X.; Li, J.; Wong, D.S.; Li, H.; You, I. Ensuring attribute privacy protection and fast decryption for outsourced data security in mobile cloud computing. *Inf. Sci.* **2017**, *379*, 42–61. [[CrossRef](#)]

39. Sun, W.; Yu, S.; Lou, W.; Hou, Y.T.; Li, H. Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. In Proceedings of the IEEE INFOCOM 2014—IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; pp. 226–234.
40. Wang, H.; Dong, X.; Cao, Z. Multi-value-Independent Ciphertext-Policy Attribute Based Encryption with Fast Keyword Search. *IEEE Trans. Serv. Comput.* **2017**, doi:10.1109/TSC.2017.2753231. [[CrossRef](#)]
41. Wang, C.; Li, W.; Li, Y.; Xu, X. A Ciphertext-Policy Attribute-Based Encryption Scheme Supporting Keyword Search Function. In *Cyberspace Safety and Security*; Springer: Cham, Switzerland, 2013; pp. 377–386.
42. Liang, K.; Susilo, W. Searchable Attribute-Based Mechanism With Efficient Data Sharing for Secure Cloud Storage. *IEEE Trans. Inf. Forensics Secur.* **2015**, *10*, 1981–1992. [[CrossRef](#)]
43. Cui, H.; Wan, Z.; Deng, R.; Wang, G.; Li, Y. Efficient and Expressive Keyword Search Over Encrypted Data in Cloud. *IEEE Trans. Dependable Secure Comput.* **2018**, *15*, 409–422. [[CrossRef](#)]
44. Wang, S.; Ye, J.; Zhang, Y. Searchable attribute-based encryption scheme with attribute revocation in cloud storage. *PLoS ONE* **2018**, *12*, e0183459. [[CrossRef](#)] [[PubMed](#)]
45. Yin, H.; Zhang, J.; Xiong, Y.; Ou, L.; Li, F.; Liao, S.; Li, K. CP-ABSE: A Ciphertext-Policy Attribute-Based Searchable Encryption Scheme. *IEEE Access* **2019**, *7*, 5682–5694. [[CrossRef](#)]
46. Lai, J.; Zhou, X.; Deng, R.H.; Chen, K. Expressive search on encrypted data. In Proceedings of the ACM Sigsac Symposium on Information, Computer and Communications Security, Hangzhou, China, 8–10 May 2013; pp. 243–252.
47. Lv, Z.; Hong, C.; Zhang, M.; Feng, D. Expressive and Secure Searchable Encryption in the Public Key Setting. In Proceedings of the International Information Security Conference, Hong Kong, China, 12–14 October 2014; pp. 364–376.
48. Peng, T.; Qin, L.; Hu, B.; Liu, J.; Zhu, J. Dynamic Keyword Search With Hierarchical Attributes in Cloud Computing. *IEEE Access* **2018**, *6*, 68948–68960. [[CrossRef](#)]
49. Zheng, Q.; Xu, S.; Ateniese, G. Vabks: Verifiable attribute-based keyword search over outsourced encrypted data. In Proceedings of the IEEE INFOCOM 2014—IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; pp. 522–530.
50. Yin, H.; Qin, Z.; Ou, L.; Li, K. A query privacy-enhanced and secure search scheme over encrypted data in cloud computing. *J. Comput. Syst. Sci.* **2017**, *90*, 14–27. [[CrossRef](#)]
51. JPBC. Available online: <http://gas.dia.unisa.it/projects/jpbc/index.html> (accessed on 10 October 2018).
52. RFC: Request for Comments Database. Available online: <http://www.ietf.org/rfc.html> (accessed on 10 October 2018).
53. Hermetic Word Frequency Counter. Available online: <http://www.hermetic.ch/wfc/wfc.htm> (accessed on 10 October 2018).