ABFKS: Attribute-Based Encryption with Functional Keyword Search in Fog Computing

Fei Meng and Mingqiang Wang*

School of Mathematics, Shandong University, Jinan Shandong 250100, China
201720214@mail.sdu.edu.cn
wangmingqiang@sdu.edu.cn

Abstract. We provide a new frame in this paper, the ciphertext-policy attribute-based encryption with functional keyword search (ABFKS) in fog computing. The ABFKS achieves functional keyword search and peer-peeping resistance, which makes it more practical and secure, while in previous attribute-based encryption with keyword search (ABKS) schemes, keyword search function is limited and ciphertext is vulnerable to man-in-the-middle attacks by adversaries who have sufficient authorities. Specifically, in ABFKS, the search process requires only one keyword, instead of each keyword as previous schemes, to be identical between the target keyword set and the ciphertext keyword set, and outputs the correlation of those two sets. Besides, the ABFKS resists peeping from peers or superiors who have the same or more attributes. Privacy and efficiency issues haven't been fully considered, therefore we provide a construction of ABFKS with privacy preserving, efficient attribute update and reverse outsourcing (ABKFS-PER). To be specific, we provide a novel method to protect the privacy of access structure by replacing each leaf node with an OR gate. In this scheme, every user has all the attributes in the cloud's view by adding fake ones so to protect the privacy of user authority. We propose a new method for attribute update, in which the key authority center only updates the user who needs update, not everyone. At last, we initially propose the concept of reverse outsourcing, namely the cloud outsourcing computational tasks to idle users to reduce its overhead.

Keywords: Attribute-based encryption · Keyword search · Fog computing · Outsourcing · Privacy preserving.

1 Introduction

Cloud computing is an Internet-based computing method, through which computing services can be provided and information can be conveniently stored or shared among various terminals and other devices on demand. With the development of 5G and Internet of Things, as well as the emergence of various intelligent devices, more and more data need to be received and processed in

* Corresponding author

the cloud, which gives rise to serious network congestion and latency. Cloud computing has been unable to meet the computing needs of the current era, so fog computing [3, 19, 20, 25] is proposed. Fog computing refers to processing data at the edge of the network. Although the overall computing ability of fog computing is not as powerful as cloud computing, it is closer to the end user, as shown in Fig. 1. So it can reduce request response time, save energy, and reduce network bandwidth. However, while enjoying outsourcing services from cloud or fog computing, data security are still the first issues to be considered.

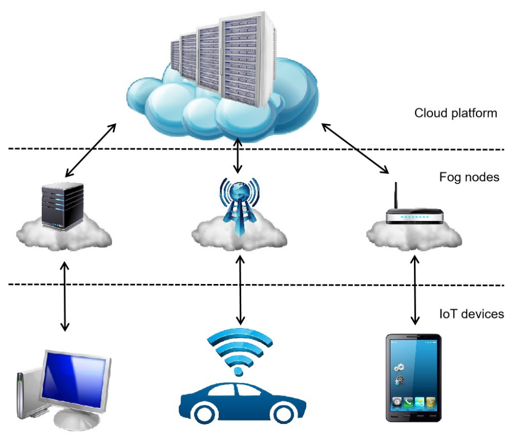

Fig. 1. The instruction of fog computing.

In order to prevent sensitive information from leaking, the data should be encrypted before uploading it to the cloud. In many application scenarios, especially in 'owner-write-users-read' application areas, the data owner needs to share data among many different users, which means that user access authorities need to be managed. Ciphertext-policy attribute-based encryption (CP-ABE) [1] can achieve fine-grained access control on encrypted data, in which the user's attributes usually represent his authority. Then, how to effectively find a ciphertext containing specific keywords among large amount of encrypted data? Searchable encryption (SE) [2, 18] allows user to search among encrypted data, in which a ciphertext keyword set is embedded, without revealing information of keywords. Moreover, ciphertext-policy attribute-based encryption with keyword search (CP-ABKS) [21, 28] supports both fine-grained access control and keyword search simultaneously, and has a wide range of applications in industrial, academic and medical fields.

Although CP-ABKS can realize both access control and keyword search, with the increase of the number of attributes and the complexity of access structure, the user's computational and storage burden will aggravate correspondingly. On this condition, outsourcing technology [8] is regarded as a good solution. Particularly, CP-ABKS schemes in fog computing environment [16, 24] reduce local computing costs by outsourcing most computational tasks to fog nodes. Users only need to perform few operations on resource-limited devices, i.e., smartphone or ipad. While searching for a ciphertext, previous CP-ABKS schemes [12, 16, 21, 24, 28] return the search results only when the target keyword set is completely identical to the ciphertext keyword set and ciphertexts are vulnerable to man-in-the-middle attacks during transmission by adversaries who have sufficient authorities. Motivated by this issue, we focus on enriching the search functionality such as result ranking, multi-keyword search [11], fuzzy keyword search [29], and providing better security guarantee.

In this paper, we propose a new frame, the ciphertext-policy attribute-based encryption with functional keyword search (ABFKS) in fog computing environment. This ABFKS achieves functional keyword search and peer-peeping resistance, which makes it more practical and secure. In specifics, when an authorized user wants to search for a ciphertext, he chooses an target keyword set to generate a trapdoor by using his secret key and sends the trapdoor to the cloud. For each ciphertext, the cloud checks his access authority with the trapdoor and picks out the accessible ciphertexts for him before search operations. Then, the cloud conducts the functional keyword search algorithm to search for the target keywords among accessible ciphertexts. This algorithm requires only one keyword between the target keyword set and the ciphertext keyword set to be identical, computes the correlation between those two sets, and ranks the accessible ciphertexts according to the correlation. While downloading an accessible ciphertext from the cloud, the user maybe eavesdropped and peeped by some malicious adversaries who have the same or more attributes as the user. The ABFKS resists this peeping attack from peers or superiors.

There are still several problems unsettled such as how to protect the privacy of access structure and user authority and how to update user's attribute efficiently. Based on the ABFKS, we present a construction of ABFKS with privacy preserving, efficient attribute update and reverse outsourcing (ABFKS-PER). To be specific, we provide a novel method to protect the privacy of access structure by replacing each leaf node with an OR gate. For each user in the ABFKS-PER, only partial secret key can be obtained, all attribute secret keys are stored in the cloud. However, this will expose user authority to the cloud. By filling up fake attribute secret keys, every user has all the attributes in the view of cloud so that the privacy of user authority is also protected. In addition, to update an attribute for a user, the key authority center only needs to generate a new secret key for him, instead of updating everyone's corresponding attribute secret key like [16, 26]. Last but not the least, there are countless idle intelligent devices connected to the Internet all over the world, which can provide computing resources for the cloud. Thus, we initially propose the concept of reverse outsourcing, namely the cloud outsourcing computational tasks to idle users to reduce its overhead. It may be a new trend in cloud computing.

1.1 Our Contributions

The main contributions of this work are shown as follows. We first present a new frame, the ciphertext-policy attribute-based encryption with functional keyword search (ABFKS) in fog computing environment. Compared with previous ABKS schemes [6, 11, 15, 16, 24], the ABFKS achieves functional keyword search and peer-peeping resistance.

- **Functional keyword search:** Many previous CP-ABKS schemes support multi-keywords search or conjunctive search [6,11,15,16]. However, their search functionality is very limited, i.e., the system returns search results only when the target keyword set is completely identical to the ciphertext keyword set. For more practical, the ABFKS achieves functional keyword search which includes result ranking, multi-keyword search and incomplete matching of keywords. Specifically, when a user issues a search query, he needs to generate a trapdoor with his own secret key and the target keywords, and send it to the cloud. While receiving a query, the cloud conducts access test and keywords matching operations successively, which means that the cloud first checks user authority and then searches for the target keywords. For a ciphertext, if the user isn't authorized to access it, the cloud will no longer conduct keyword matching for it, but turn to the next one directly. For multi-keyword search in massive encrypted data, this simple change can save considerable computational costs of the cloud. A ciphertext is regarded as an accessible ciphertext for a user, as long as he is authorized to access it and there is at least one keyword matched successfully in the target keyword set. Finally, the cloud returns all accessible ciphertexts and their correlation coefficients, each of which reflects the correlation between the ciphertext and the target keywords. For convenience, the cloud ranks the them in ascending or descending order according to the correlation coefficient. Throughout the search process, no sensitive information about target keywords or ciphertext keywords is leaked.
- **Peer-peeping resistance:** In previous schemes [16, 21, 28], provided that someone with corresponding attributes or authority, obtains a ciphertext, he can decrypt it. So, ciphertexts are vulnerable to man-in-the-middle attacks by this kind of adversaries who have sufficient authorities. For example, while downloading a ciphertext, an employee maybe eavesdropped and peeped by his colleagues or bosses with the same or even higher access authority. On this condition, his privacy will be revealed to peers or superiors. For this reason, The ABKFS generates the keyword ciphertext, instead of merely an index, as well as the attribute ciphertext to encrypt data. In ABFKS system, anyone is able to decrypt a ciphertext, if and only if he has sufficient authority and knows at least one element in the ciphertext keyword set simultaneously. Therefore, even if a malicious adversary eavesdrops a ciphertext and has the corresponding authority, he still can not decrypt it.

Though the ABFKS is more practical and secure compared with previous schemes. Privacy and efficiency issues are still not fully considered. In order to protect the privacy of access structure and user authority and achieve efficient attribute update, we further provide a construction of ABFKS with privacy preserving, efficient attribute update and reverse outsourcing (ABFKS-PER).

- **Privacy-preserving**: In traditional ABE schemes [1], the access structure, which defines an access policy, is sent along with the ciphertext. This property is not suitable when the access policy contains some sensitive information. Moreover in many ABKS schemes [12, 16, 21, 24, 28], the cloud is required to check user's authority so that the attributes of each user are exposed to the cloud, which leaks the privacy of user. Hence, We provide a novel method to preserve the privacy of access policy and user authority against the cloud. For an access structure, we transform it into a new access structure by replacing each leaf node with an OR gate. This OR gate has two child nodes, one of which is the same as the replaced leaf node, and the other is randomly selected from nodes disjoint with the original access structure. Therefore, if there are n leaf nodes in the original access structure, the probability of the cloud to recover it from the new one is 2^{-n} so that privacy of access structure is preserved. For each system user, only partial secret key is stored locally, all attribute secret keys are randomized and stored in the cloud. By filling up fake attribute secret keys, every system user has all the attributes in the view of the cloud and the cloud has no ability to identify the real keys. As a result, the privacy of user authority is also preserved.

- **Efficient attribute update**: Zhang et al. [26] propose the first CP-ABE scheme with attribute update for fog computing. When a user wants to update an attribute, the key authority center has to update every user and every ciphertext associated with this attribute, whether those users have applied for attribute update or not. If there are millions of users in the system, this method will no longer be applicable. In the ABFKS-PER, user secret key is divided into two parts. Partial secret key is sent to the user, while all attribute secret keys are randomized and stored in the cloud. To update an attribute for a user, the key authority center only needs to regenerate a new secret key for him. Since the user does't have attribute secret key, it is not necessary to update the corresponding attribute secret key of every user, nor the associated ciphertexts as [16, 26].
- **Reverse outsourcing**: It is generally known that the cloud service provider can provide computing services for end users to reduce their local computational burden. However, little attention has been paid to reduce computing pressure of the cloud. There are countless intelligent devices connected to the Internet all over the world. They have certain computing power and are idle most of the time. These computing resources can be aggregated to provide computing services to the cloud. Thus, we initially propose this interesting concept of reverse outsourcing, namely the cloud outsourcing computational tasks to idle users to reduce its overhead. In addition, we define the rational idle user model, and analyze the best strategy for the user in this model by the Nash equilibrium theory. We think reverse outsourcing may be a new trend in cloud computing.

1.2 Organization

This paper is organized as follows. Section 2 discusses several previous works. Section 3 describes the necessary preliminaries. Section 4 presents the system and security model. We give a concrete construction and explicit analysis of ABKFS in section 5 and section 6 respectively. In section 7, we improve introduce the construction and analysis of ABFKS-PER. In the end, section 8 summarizes the paper and prospects for the future research.

2 Related Works

Sahai et al. [17] initially introduced the concept of ABE. Generally, there are two types of ABE schemes, i.e., key-policy ABE (KP-ABE) [7] and ciphertext-policy ABE (CP-ABE) [1]. Bethencourt et al. [1] proposed the first CP-ABE scheme which realizes fine-grained access control based on the tree-based structure. From then on, large numbers of CP-ABE schemes have been proposed to achieve various functions. Cheung et al. [5] proposed a CP-ABE scheme based on the AND gate access structure. Horvath et al. [10] proposed a multi-authority CP-ABE scheme with identity-based revocation. Wang et al. [23] devised a CP-ABE scheme with hierarchical data sharing. However, with the increase of the number of attributes and the complexity of access structure, general CP-ABE schemes are computationally expensive.

Through outsourcing technology [8], the computational and storage burden of users can be outsourced to some third parties. Green et al. [9] provided a method to outsource the decryption of ABE ciphertexts. Li et al. [13] outsources both key-issuing and decryption. Zhang et al. [27] fully outsources key generation, encryption and decryption. In the wake of 5G and IoT techniques, fog computing is considered as a new data resource, which can provide many high-quality outsourcing services. Zuo et al. [30] proposed a practical CP-ABE scheme in fog computing environment and Zhang et al. [26] initially supports fog computing as well as attribute update.

Searching over encrypted data, the keyword can not be revealed because it may reflect sensitive information of ciphertext. In 2000, Song et al. [18] initially introduced a searchable encryption (SE) technique. Boneh et al. [2] proposed the first public key encryption with keyword search. After that, various SE schemes have been proposed one after another, which makes the search function more and more abundant, such as single keyword search [22], multi-keyword search [4] and fuzzy keyword search [14]. Besides, plenty of ciphertext-policy attribute-based encryption (CP-ABKS) schemes [6, 11, 12, 15, 21, 28] support both fine-grained access control and keyword search simultaneously. Furthermore, many of the latest CP-ABKS schemes [16, 24] are constructed in fog computing environment.

3 Preliminaries

In this section, we introduce some background knowledge, which includes access structure, access tree, bilinear maps, Diffie-Hellman assumption and its variants.

3.1 Access Structures

Definition 1 (Access structure [1]). *Let $\{P_1, P_2, \ldots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1,P_2,\ldots,P_n\}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \ldots, P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1,P_2,\ldots,P_n\}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.*

In this paper, attributes take the role of the parties and we only focus on the monotone access structure $\mathbb{A}$, which consists of the authorized sets of attributes. Obviously, attributes can directly reflect a user's authority.

Definition 2 (Access tree [1]). *Let $\mathcal{T}$ be a tree representing an access structure. Each non-leaf node of the tree represents a threshold gate, described by its children and a threshold value. If num_x is the number of children of a node x and k_x is its threshold value, then $0 \leq k_x \leq num_x$. When $k_x = 1$, the threshold gate is an OR gate and when $k_x = num_x$, it is an AND gate. Each leaf node x of the tree is describe by an attribute and a threshold value $k_x = 1$.*

To facilitate working with the access tree, we define a few functions. We denote the parent of the node x in the tree by $parent(x)$. The function $att(x)$ is defined only if x is a leaf node and denotes the attribute associated with the leaf node x in the tree. The access tree $\mathcal{T}$ also defines an ordering between the children of every node, that is, the children of a node are numbered from 1 to num. The function $index(x)$ returns such a number associated with the node x. Where the index values are uniquely assigned to nodes in the access structure for a given key in an arbitrary manner.

Definition 3 (Satisfying an access tree [1]). *Let $\mathcal{T}$ be an access tree with root r. Denote by $\mathcal{T}_x$ the subtree of $\mathcal{T}$ rooted at the node x. Hence $\mathcal{T}$ is the same as $\mathcal{T}_r$. If a set of attributes satisfies the access tree $\mathcal{T}_x$, we denote it as $\mathcal{T}_x(\gamma) = 1$. We compute $\mathcal{T}_x(\gamma)$ recursively as follows. If x is a non-leaf node, evaluate $\mathcal{T}_{x'}(\gamma) = 1$ for all children x' of node x. $\mathcal{T}_x(\gamma)$ returns 1 if and only if at least k_x children return 1. If x is a leaf node, then $\mathcal{T}_x(\gamma)$ returns 1 if and only if $att(x) \in \gamma$.*

3.2 Bilinear Maps and DDH Assumptions

As in [1]. We introduce some useful facts about bilinear maps. Let $\mathbb{G}_0$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}_0$ and e be a efficient computable bilinear map, $e : \mathbb{G}_0 \times \mathbb{G}_0 \longrightarrow \mathbb{G}_T$. The bilinear map e has a few properties: (1) Bilinearity: for all $u, v \in \mathbb{G}_0$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$. (2) Non-degeneracy: $e(g, g) \neq 1$. We say that $\mathbb{G}_0$ is a bilinear group if the group operation in $\mathbb{G}_0$ and the bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_0 \longrightarrow \mathbb{G}_T$ are both efficiently computable. Notice that the map e is symmetric since $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$. We briefly recall the definitions of the decisional Diffie-Hellman (DDH) assumption and its varieties as follows.

Definition 4 (DDH). *Let $\mathcal{G}$ be a an algorithm that takes as input a security parameter λ and outputs a tuple $\mathbb{G} = (p, G, g)$ where p is a prime, G is a cyclic group of order p, and g is a generator of G. For any PPT algorithm $\mathcal{A}$, there exists a negligible function $negl(\cdot)$, such that*

$$\Big| \Pr\left[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1\right] - \Pr\left[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1\right] \Big| \leq negl(\lambda), \quad (1)$$

where $\mathbb{G} = (p, G, g) \leftarrow \mathcal{G}(\lambda)$, and $x, y, z \leftarrow \mathbb{Z}_p$ are uniform and independent.

It's easy to verify that the following lemma is correct by hybrid experiments.

Lemma 1 (t-DDH). *For any positive integer t and any PPT algorithm $\mathcal{A}$, we have $\Big| \Pr\left[\mathcal{A}\left(\mathbb{G}, q, g, g^x, \{g^{y_i}, g^{z_i} \mid \forall i \in [1,t]\}\right) = 1\right] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, \{g^{y_i}, g^{xy_i} \mid \forall i \in [1,t]\}) = 1] \Big| \leq negl(\lambda)$ for some negligible function $negl(\cdot)$, where x, y_i, z_i are selected randomly from $\mathbb{Z}_p$ for each $i \in [1,t]$.*

As in [26], the decisional bilinear Diffie-Hellman (DBDH) problem is defined as follows.

Definition 5 (DBDH). *Let $\mathbb{G}_0, \mathbb{G}_T$ are multiplicative cyclic groups with prime order p according to a security parameter λ and the generator of $\mathbb{G}_0$ is g. Let $e : \mathbb{G}_0 \times \mathbb{G}_0 \longrightarrow \mathbb{G}_T$ be a bilinear map, $x, y, z \in \mathbb{Z}_p$ and $R \in \mathbb{G}_T$ are selected randomly. For any PPT algorithm $\mathcal{A}$, there exists a negligible function $negl(\cdot)$, such that*

$$\Big| \Pr[\mathcal{A}(g, g^x, g^y, g^z, Z = e(g,g)^{xyz}) = 1] - \Pr[\mathcal{A}(g, g^x, g^y, g^z, Z = R) = 1] \Big| \leq negl(\lambda). \quad (2)$$

4 System and Security Model

4.1 System Description

The ABFKS mainly consists of five entities i.e., Key Authority Center (KAC), Data Owner (DO), Cloud Server (CS), User (U), and Fog Nodes, which are shown in Fig. 2.

- Key Authority Center (KAC): The KAC is a fully trusted third party which is in charge of generating public parameters and replying secret key to each authorized user as well as handling attribute update.
- Data Owner (DO): The DO defines an access structure, selects a set of keywords to generate a ciphertext CT with the help of fog nodes, then uploads CT to the CS.
- Cloud Server (CS): The CS has powerful computation and huge storage capacities, which provides computing and storage service.
- User (U): The U is constrained by limited resources. However, he can search and decrypt according to his authority with the help of fog nodes.
- Fog Nodes: Fog nodes can help the DO or the U to reduce computational overhead during encryption or trapdoor generation and decryption.

4.2 System Overview

The overview of ABFKS scheme is shown as follows:

- **Setup**$(1^\lambda, \mathcal{L})$: Given security parameter λ and a set of all possible attributes $\mathcal{L}$, the KAC generates public key PK and master key MSK.
- **KeyGen**(MSK, S): The KAC uses MSK to generate a secret key SK according to the user's attribute set S.
- **Enc**$(PK, \mathcal{T}, M, KW)$: The DO defines a access structure $\mathcal{T}$ and selects a set of keyword KW to encrypt M with the help of fog nodes, then uploads the ciphertext to the CS.
- **Trap**(SK, KW'): To issue a search query, the U generates a trapdoor (Tc, Tk) by his own secret key SK and a set of target keyword KW' with the help of fog nodes.
- **Search**(CT, Tc, Tk): Given a trapdoor (Tc, Tk), the CS can conduct access test and keyword matching operations, and return relevant results.
- **Dec**(CT, SK): The U can download and decrypt the accessible ciphertext according to the relevance between KW and KW' with the help of fog nodes.

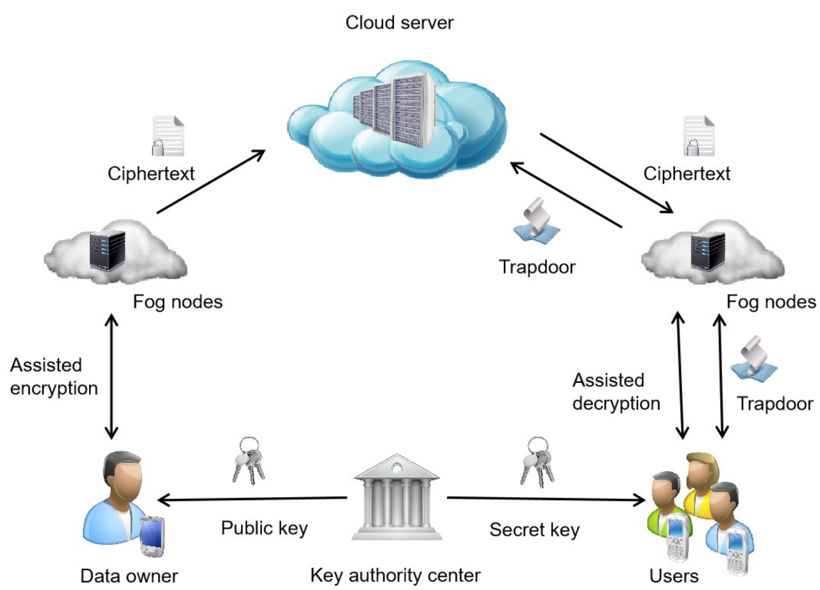

Fig. 2. System descryption of fog computing.

4.3 Threat Model

in this paper, we assume that the KAC is a fully trusted third party, while the CS and fog nodes are honest-but-curious entities, which exactly follow the protocol specifications but also are curious about the sensitive information of ciphertexts and trapdoors. Users are not allowed to collude with CS or fog nodes.

Nevertheless, malicious users may collude with each other to access some unauthorized ciphertexts. While downloading a ciphertext, the user may be eavesdropped and peeped by some adversaries who have the same or more attributes, such as his peers or superiors.

4.4 Security Model

We define the chosen plaintext security of ABFKS scheme. The security game is as follows

- *Initialization*: A PPT $\mathcal{A}$ chooses a challenge access tree $\mathcal{T}^*$ and dispatches it to challenger $\mathcal{C}$.
- *Setup*: $\mathcal{C}$ runs **Setup** algorithm and return public key PK to $\mathcal{A}$.
- *Phase* 1: $\mathcal{A}$ adaptively chooses an attribute set S which doesn't satisfy $\mathcal{T}^*$, and submits it to $\mathcal{C}$ asking for a secret key SK corresponding with S. In response to secret key query from $\mathcal{A}$, $\mathcal{C}$ runs **KeyGen** algorithm and returns SK to $\mathcal{A}$.
- *Challenge*: $\mathcal{A}$ chooses two challenge message m_0, m_1 with a set of keywords KW^* and submits them to $\mathcal{C}$ to be challenged. $\mathcal{C}$ picks a random bit $\vartheta \in \{0, 1\}$ and runs **Enc** algorithm to encrypt m_ϑ. Afterwards, $\mathcal{C}$ returns the challenge ciphertext CT^* to $\mathcal{A}$.
- *Phase* 2: This phase is similar to Phase 1.
- *Guess*: $\mathcal{A}$ picks a guess bit ϑ' of ϑ. If and only if $\vartheta' = \vartheta$, $\mathcal{A}$ wins out, otherwise, it loses the game. Then, $\mathcal{A}'s$ advantage to win this security game is defined as $Adv(\mathcal{A}) = \left| Pr[\vartheta' = \vartheta] - \frac{1}{2} \right| \leq \epsilon$.

Definition 6. *ABFKS scheme can achieve CPA security if there exist no PPT adversary which can break the above security game with a non-negligible advantage ϵ under the DBDH assumption.*

In addition, ABFKS scheme also achieves chosen keyword security as defined in the following security game.

- *Initialization*: $\mathcal{A}$ selects two different challenge keyword sets KW^{0*} and KW^{1*}, each of which contains t keywords in total. $\mathcal{A}$ sends them to challenger $\mathcal{C}$.
- *Setup*: $\mathcal{C}$ runs **Setup** algorithm and publishes public parameters.
- *Phase* 1: $\mathcal{A}$ adaptively queries $\mathcal{C}$ for a partial trapdoor Tk of KW which is unequal to KW^{0*} or KW^{1*}. In response, $\mathcal{C}$ runs **Trap** then responds $\mathcal{A}$ with Tk.
- *Challenge*: Given challenge keyword sets KW^{0*} and KW^{1*}, $\mathcal{C}$ picks a random bit $\vartheta \in \{0, 1\}$ and runs **Enc** algorithm to generate partial ciphertext CT_2^*
- *Phase* 2: This phase is similar to Phase 1.
- *Guess*: $\mathcal{A}$ picks a guess bit ϑ' of ϑ. If and only if $\vartheta' = \vartheta$, $\mathcal{A}$ wins out, otherwise, it loses the game. Then, $\mathcal{A}'s$ advantage to win this security game is defined as $Adv(\mathcal{A}) = \left| Pr[\vartheta' = \vartheta] - \frac{1}{2} \right| \leq \epsilon$.

Definition 7. *ABFKS scheme can achieve CKA security if there exist no PPT adversary which can break the above security game with a non-negligible advantage ϵ under the t-DDH assumption.*

As a supplement, We initially define the peer-peeping resistance as a new security requirement.

Definition 8 (Peer-peeping resistance). *Assume that the U is authorized to access some ciphertext CT stored in the CS. An adversary $\mathcal{A}$ may eavesdrop the CT while the U is downloading it. Usually, if $\mathcal{A}$ has the same or higher authority as U, i.e., a peer or superior, the CT will be decrypted and peeped by $\mathcal{A}$. However, peer-peeping resistance is a new security requirement, which requires that even if the ciphertext is eavesdropped by some authorized adversaries, it still cannot be decrypted and peeped.*

5 Construction of ABFKS Scheme

In this section, we present a concrete construction of ABFKS scheme. Without loss of generality, we suppose that there are n possible attributes in total and $\mathcal{L}$ is a set of all possible attributes, where $\mathcal{L} = \{a_1, a_2, ..., a_n\}$. Assume $\mathbb{G}_0, \mathbb{G}_T$ are multiplicative cyclic groups with prime order p and the generator of $\mathbb{G}_0$ is g. λ is a security parameter which determines the size of groups. Moreover, let $e : \mathbb{G}_0 \times \mathbb{G}_0 \longrightarrow \mathbb{G}_T$ be a bilinear map. Let $H : \{0,1\}^* \longrightarrow \mathbb{G}_0$ be a hash function which maps any string to a random element in $\mathbb{Z}_p$. We also define the Lagrange coefficient $\Delta_{i,S}(x) = \Pi_{j \in S, j \neq i} \frac{x-j}{i-j}$, where $i \in \mathbb{Z}_p$ and S is a set composed of elements in $\mathbb{Z}_p$. The details of our scheme are as follows.

- **Setup**$(1^\lambda, \mathcal{L}) \rightarrow (PK, MSK)$: Given a security parameter λ and a set of all possible attributes $\mathcal{L}$, the Key Authority Center (KAC) chooses a bilinear group $\mathbb{G}_0$ with prime order p and generator g. Next, it randomly picks out $\alpha, \beta \in \mathbb{Z}_p$ and $h \in \mathbb{G}_0$. For each attribute $a_j \in \mathcal{L}$, it selects a random v_j and computes $PK_j = g^{v_j}$. Finally, it generates the master key MSK and publishes the public key PK.

$$PK = \left\{\mathbb{G}_0, g, h, g^\alpha, h^\beta, e(g,g)^\beta, \{PK_j = g^{v_j} \mid \forall a_j \in \mathcal{L}\}\right\}; \tag{3}$$

$$MSK = \{\alpha, \beta, \{v_j \mid \forall a_j \in \mathcal{L}\}\}. \tag{4}$$

- **KeyGen**$(MSK, S) \rightarrow SK$: While receiving an attribute set S from the U, the KAC selects $r, r' \in \mathbb{Z}_p$ at random and generates a secret key SK, then sends it back to the U in secret channel.

$$SK = \left\{\beta + \alpha r, g^{\alpha r} h^{r'}, g^{r'}, \{g^{\frac{\alpha r}{v_j}}, h^{\frac{\alpha r}{v_j}} \mid \forall a_j \in S\}\right\}. \tag{5}$$

- **Enc**$(PK, \mathcal{T}, M, KW) \rightarrow CT$: The DO chooses a random ck as a symmetric encryption key and encrypts message M with ck by using symmetric encryption such as AES, namely $E_{ck}(M)$. In order to encrypt ck, the whole encryption procession consists of the follow steps.
  1. **Attribute Ciphertext**: The DO sends an access tree $\mathcal{T}$ to fog nodes, which describes an access policy. Fog nodes randomly chooses a polynomial q_x for each node x of $\mathcal{T}$ from the root node R in a top-down manner. For each node x of $\mathcal{T}$, $d_x = k_x - 1$, where d_x is the degree of q_x and k_x is

the threshold value of x. Beginning with root node R, fog nodes pick a random $s_1 \in \mathbb{Z}_p$ and set $q_R(0) = s_1$. Next, they randomly choose d_R other points of q_R to define the polynomial completely. For any other node x, fog nodes set $q_x(0) = q_{parent(x)}(index(x))$ and choose d_x other point to define q_x completely. Let $\mathcal{X}$ be a set of attributes corresponding with all leaf nodes in $\mathcal{T}$. Fog nodes construct the attribute ciphertext CT_1' and send it back to the DO.

$$CT_1' = \left\{\mathcal{T}, g^{s_1}, h^{s_1}, h^{\beta s_1}, \{C_j = g^{v_j q_x(0)} \mid \forall a_j = att(x) \in \mathcal{X}\}\right\}. \quad (6)$$

2. **Keyword Ciphertext**: Assume that the DO chooses t different keywords associated with M in total and the keyword set $KW = \{kw_1, kw_2, \ldots, kw_t\}$. Then, the DO randomly selects $r_1, r_2 \in \mathbb{Z}_p$, computes $h^{\frac{1}{r_1}}$, KW_1 and KW_2 and sends KW_1 and KW_2 to fog nodes, where

$$KW_1 = \{H(kw_1)r_1, H(kw_2)r_1, \ldots, H(kw_t)r_1\}; \quad (7)$$
$$KW_2 = \{H(kw_1)r_2, H(kw_2)r_2, \ldots, H(kw_t)r_2\}. \quad (8)$$

After receiving KW_1 and KW_2 from the DO, fog nodes compute the keyword ciphertext CT_2' and send it back to the DO.

$$CT_2' = \{C_1^i = g^{\frac{1}{H(kw_i)r_1}}, C_2^i = g^{\frac{1}{H(kw_i)r_2}} \mid \forall i \in [1, t]\}. \quad (9)$$

3. The DO picks a random $s_2 \in \mathbb{Z}_p$ and generates CT_1 and CT_2 with CT_1' and CT_2' as

$$CT_1 = \{g^{s_2}, g^{s_1} g^{s_2}, h^{s_1} h^{s_2}, CT_1'\}; \quad CT_2 = \{g^{\frac{1}{r_1}}, CT_2'\}. \quad (10)$$

Then, it computes $e(g,g)^{\beta s_2}$ and $e(g,g)^{\frac{1}{r_2}}$ to get the final ciphertext CT, where

$$CT = \{\mathcal{T}, E_{ck}(M), C = ck \cdot e(g,g)^{\beta s_2} \cdot e(g,g)^{\frac{1}{r_2}}, CT_1, CT_2\}. \quad (11)$$

- **Trap**$(SK, KW') \rightarrow (Ta, Tk)$: When the U wants search for a set of target keywords KW' in the CS, he can generate a trapdoor by the secret key SK and KW' with the help of fog nodes. Supposed that there are t different keywords in KW', i.e., $KW' = \{kw_1', kw_2', \ldots, kw_t'\}$.

1. **Attribute Trapdoor**: The U selects a random $d_1, r_3 \in \mathbb{Z}_p$, and computes the attribute trapdoor Ta with SK as

$$Ta = (Ta_0, Ta_1, Ta_2) = (g^{\frac{1}{d_1}}, \frac{\beta + \alpha r}{d_1}, \{Ta_2^j = h^{\frac{\alpha r}{v_j d_1}} \mid \forall a_j \in S\}). \quad (12)$$

2. **Keyword Trapdoor**: The U computes g^{r_3}, KW_1', and sends KW_1' to fog nodes, where $KW_1' = \{H(kw_1')r_3, H(kw_2')r_3, \ldots, H(kw_t')r_3\}$. The fog nodes compute KW_2' and sent it back to the U, where $KW_2' = \{g^{\frac{1}{H(kw_j')r_3}} \mid \forall j \in [1, t]\}$. Finally, the U can generate the keyword trapdoor Tk as

$$Tk = (Tk_0, Tk_1) = (g^{r_3}, \{Tk_1^j = g^{\frac{1}{H(kw_j')r_3}} \mid \forall j \in [1, t]\}). \quad (13)$$

- **Search**$(CT, Ta, Tk) \rightarrow \perp or\, l \in [1, t]$: The CS has to check whether the trapdoor from U is available or not. Only when U is authorized to access CT and there is at least one keyword to be identical in KW and KW', the search algorithm outputs $l \in [1, t]$, i.e., there are l keywords in common between KW and KW'. Otherwise, the algorithm outputs $\perp$. The entire algorithm consists of two parts: access test and keyword matching.
  1. **Access Test**: For each node x of $\mathcal{T}$ in CT, the CS runs a recursive algorithm as follows.
     (1) If x is a leaf node of $\mathcal{T}$. Let $a_j = att(x)$. If $a_j \notin S$, $F'_x(C_j, Ta_2^j, x) = null$. If $a_j \in S$, then the CS computes
     $$F'_x(C_j, Ta_2^j, x) = e(g^{v_j q_x(0)}, h^{\frac{\alpha r}{v_j d_1}}) = e(g, h)^{\frac{\alpha r q_x(0)}{d_1}}. \tag{14}$$
     (2) If x is a non-leaf node, for all child nodes z of x, the CS runs $F'_z = F'_z(C_i, Ta_2^i, z)$ recursively. Let S_x be an arbitrary $k_x - sized$ set of z, and satisfying $F'_z \neq null$. If S_x doesn't exist, $F'_x = null$. Otherwise, the CS calculates
     $$F'_x = \prod_{z \in S_x} F_z'^{\Delta_{i,S'_x}(0)} \tag{15}$$
     $$= \prod_{z \in S_x} (e(g, h)^{\alpha r q_z(0)})^{\Delta_{i,S'_x}(0)} \tag{16}$$
     $$= \prod_{z \in S_x} (e(g, h)^{\alpha r q_{parent(z)}(index(z))})^{\Delta_{i,S'_x}(0)} \tag{17}$$
     $$= e(g, h)^{\frac{\alpha r q_x(0)}{d_1}}, \tag{18}$$
     where $i = index(z)$ and $S'_x = \{index(z) : z \in S_x\}$.
     By calling the above functions on the root node R of $\mathcal{T}$, the CS can checks the whether following equation holds or not.
     $$F'_R \cdot e(Ta_0, h^{\beta s_1}) = e(g^{s_1}, h)^{Ta_1}. \tag{19}$$
     If S satisfies $\mathcal{T}$, which means U is authorized to access CT, the CS computes $F'_R = e(g, h)^{\frac{\alpha r s_1}{d_1}}$ and the above equation holds, namely
     $$F'_R \cdot e(Ta_0, h^{\beta s_1}) = e(g, h)^{\frac{\alpha r s_1}{d_1}} \cdot e(g^{\frac{1}{d_1}}, h^{\beta s_1}) = e(g^{s_1}, h)^{Ta_1}. \tag{20}$$
     Then, the CS continues to match the keywords between KW and KW'. Otherwise, the CS no longer conducts the following keyword matching operations for this ciphertext and turns to the next one.
  2. **Keyword Matching**: The CS receives $Tk = (Tk_0, TK_1)$ from U to construct the keyword search vector $\overrightarrow{Tk_1}$ and uses CT to construct the keyword ciphertext vector $\overrightarrow{C_1^i}$, where
     $$\overrightarrow{Tk_1} = (Tk_1^1, Tk_1^2, \ldots, Tk_1^t), \quad Tk_1^j = g^{H(kw'_j)r_3}; \tag{21}$$
     $$\overrightarrow{C_1^i} = (C_1^1, C_1^2, \ldots, C_1^t), \quad C_1^i = g^{\frac{1}{H(kw_i)r_1}}. \tag{22}$$

Next, the CS interacts with $\overrightarrow{Tk_1}$ and $\overrightarrow{C_1^i}$ to perform the keyword matching. For each $j \in [1,t]$, the CS examine whether there is an $i \in [1,t]$ that makes the following equation valid.

$$e(Tk_1^j, C_1^i) = e(Tk_0, g^{\frac{1}{r_1}}) \tag{23}$$

(1) If for a j, no i makes the above formula hold, the CS outputs a matching result $mr_j = 0$, which means that kw'_j in KW' is not in KW.
(2) If the above formula is established, it means that for k'_j there exists a k_i such that $k'_j = k_i$, i.e.,

$$e(Tk_1^j, C_1^i) = (e(g,g)^{\frac{r_3}{r_1}})^{\frac{H(kw'_j)}{H(kw_i)}} = e(Tk_0, g^{\frac{1}{r_1}}). \tag{24}$$

Then, the CS sets $mr_j = i$, which means the j^{th} keyword of KW' is equal to the i^{th} keyword in KW.
After matching each keyword, the CS constructs a matching result vector $\overrightarrow{mr}$, where

$$\overrightarrow{mr} = (mr_1, mr_2, \ldots, mr_t), \tag{25}$$

and computes the hamming weight of $\overrightarrow{mr}$, i.e., $l = \mathcal{H}_m(\overrightarrow{mr}) \in [0,t]$, which reflects the correlation between KW' and KW. If and only if the U has passed access test and $l > 0$, the **Search** algorithm outputs $l \in [1,t]$ and the U is allowed to access CT. Otherwise $\perp$ and turns to the next ciphertext.

Ultimately, the cloud generates the following Table 1 for U and sorts it in descending order according to the correlation, supposed that there are N ciphertexts in total for U to access.

Table 1. Access table for U generated by the search algorithm.

Accessible ciphertext	Matching result vector	Correlation
$CT_{(1)}$	$\overrightarrow{mr}_{(1)}$	$l_{(1)}$
$CT_{(2)}$	$\overrightarrow{mr}_{(2)}$	$l_{(2)}$
$\vdots$	$\vdots$	$\vdots$
$CT_{(N)}$	$\overrightarrow{mr}_{(N)}$	$l_{(N)}$

- **Dec**$(CT, SK) \to ck$: The U can access CT in Table 1 according to l, and decrypt it with the help of fog nodes in the following steps.
  1. The U selects a random $d_2 \in \mathbb{Z}_p$, keeps it secret, and computes a random secret key SK' as

$$SK' = \left\{SK'_1, SK'_2, SK'_3, SK'_4 = \{SK'^j_4\}\right\} \tag{26}$$

$$= \left\{g^{\frac{\beta+\alpha r}{d_2}}, g^{\frac{\alpha r}{d_2}} h^{\frac{r'}{d_2}}, g^{\frac{r'}{d_2}}, \{g^{\frac{\alpha r}{v_j d_2}} \mid \forall a_j \in S\}\right\}, \tag{27}$$

then sends SK' to fog nodes.

2. The fog nodes interact SK' and CT to perform some precomputation, which greatly reduces the computational cost of user decryption. The interaction process is similar to the access test algorithm described above. For each node x of $\mathcal{T}$ in CT, the fog nodes perform the following recursive algorithm.
(1) If x is a leaf node of $\mathcal{T}$. Let $a_j = att(x)$. If $a_j \notin S$, $F_x(C_j, SK_4'^j, x) = null$. If $a_j \in S$, then fog nodes compute

$$F_x(C_j, SK_4'^j, x) = e(g^{v_j q_x(0)}, g^{\frac{\alpha r}{v_j d_2}}) = e(g,g)^{\frac{\alpha r q_x(0)}{d_2}}. \quad (28)$$

(2) If x is a non-leaf node, for all child nodes z of x, the fog nodes calculate $F_z = F_z(C_i, SK_4'^i, z)$ recursively. Let S_x be an arbitrary $k_x - sized$ set of z, and satisfying $F_z \neq null$. If S_x doesn't exist, $F_x = null$. Otherwise, the fog nodes compute

$$F_x = \prod_{z \in S_x} F_z^{\Delta_{i,S_x'}(0)} = e(g,g)^{\frac{\alpha r q_x(0)}{d_2}}. \quad (29)$$

By running the above algorithm recursively, the fog nodes obtain $F_R = e(g,g)^{\frac{\alpha r s_1}{d_2}}$ for the root node R of $\mathcal{T}$ and continue to compute

$$A = \frac{e(SK_2', g^{s_1} g^{s_2})}{e(SK_3', h^{s_1} h^{s_2})} = \frac{e(g^{\frac{\alpha r}{d_2}} h^{\frac{r'}{d_2}}, g^{s_1} g^{s_2})}{e(g^{\frac{r'}{d_2}}, h^{s_1} h^{s_2})} = e(g,g)^{\frac{\alpha r (s_1+s_2)}{d_2}} \quad (30)$$

and

$$B = e(SK_1', g^{s_2}) = e(g^{\frac{\beta + \alpha r}{d_2}}, g^{s_2}) = e(g,g)^{\frac{\beta s_2}{d_2}} \cdot e(g,g)^{\frac{\alpha r s_2}{d_2}}, \quad (31)$$

to calculate

$$D = \frac{B \cdot F_R}{A} = e(g,g)^{\frac{\beta s_2}{d_2}}. \quad (32)$$

Then, fog nodes return D to the U.
3. The U obtains $\overrightarrow{mr}$ in Table 1. There is at least one mr_j in $\overrightarrow{mr}$ such that $mr_j = i \neq 0$, i.e., $kw_j' = kw_i$. Thus, the U can make use of the j^{th} keyword in KW' and i^{th} keyword in KW to compute E as

$$E = e(g^{H(kw_j')}, C_2^i) = e(g^{H(kw_j')}, g^{\frac{1}{H(kw_i) r_2}}) = e(g,g)^{\frac{1}{r_2}}. \quad (33)$$

Finally, the U derives ck as

$$\frac{C}{D^{d_2} \cdot E} = \frac{ck \cdot e(g,g)^{\beta s_2} \cdot e(g,g)^{\frac{1}{r_2}}}{(e(g,g)^{\frac{\beta s_2}{d_2}})^{d_2} \cdot e(g,g)^{\frac{1}{r_2}}} = ck. \quad (34)$$

So far, the U can decrypt $E_{ck}(M)$ with ck by symmetric decryption.

- **Attribute update**: We can use the method of [26] to keep our ABFKS scheme with attribute update, which is very important to protect data from eavesdropping and sniffing by revoked users. Next, we briefly describe the basic idea as follows.

1. To update $a_j \rightarrow a_w$, the KAC selects a random $v'_j \in \mathbb{Z}_p$ and computes $uk_{j\rightarrow w} = \frac{v_j}{v_w}$, $uk_{j\rightarrow j} = \frac{v_j}{v'_j}$, and $cu_{j\rightarrow j} = \frac{v'_j}{v_j}$. The KAC updates $PK_{j\rightarrow j} = PK_j^{uk_{j\rightarrow j}} = g^{v'_j}$ and respectively sends $uk_{j\rightarrow w}, uk_{j\rightarrow j}, cu_{j\rightarrow j}$ to updated users, non-updated users, and the CS.
2. The updated user updates their secret key as

$$SK = \left\{\beta + \alpha r, g^{\alpha r} h^{r'}, g^{r'}, \{g^{\frac{\alpha r}{v_i}}, h^{\frac{\alpha r}{v_i}} \mid \forall a_i \in S \setminus \{a_j\}\}, g^{\frac{\alpha r}{v_w}}, h^{\frac{\alpha r}{v_w}}\right\}; \quad (35)$$

   Similarly, non-updated users update their secret keys as

$$SK = \left\{\beta + \alpha r, g^{\alpha r} h^{r'}, g^{r'}, \{g^{\frac{\alpha r}{v_i}}, h^{\frac{\alpha r}{v_i}} \mid \forall a_i \in S \setminus \{a_j\}\}, g^{\frac{\alpha r}{v'_j}}, h^{\frac{\alpha r}{v'_j}}\right\}. \quad (36)$$

3. In addition, the CS updates the ciphertext CT'_1 in CT as

$$CT'_1 = \left\{ \begin{array}{l} \mathcal{T}, g^{s_1}, h^{s_1}, h^{\beta s_1}, C_j = g^{v'_j q_x(0)}, \\ \{C_i = g^{v_i q_x(0)} \mid \forall a_i = att(x) \in \mathcal{X} \setminus \{a_j\}\} \end{array} \right\}. \quad (37)$$

6 Analysis of ABFKS

In this section, we provide a security analysis of ABFKS scheme, and then compare its function and efficiency with other schemes.

6.1 Security Analysis

Here, we prove the IND-CPA and IND-CKA security of our ABFKS scheme formally and then discuss the peer-peeping resistance as a supplement.

Theorem 1. *Supposed that a PPT adversary $\mathcal{A}$ can break the IND-CPA security of our ABFKS scheme with a non-negligible advantage $\epsilon > 0$, then a PPT simulator $\mathcal{B}$ can be constructed to distinguish a DBDH tuple from a random tuple with an advantage $\frac{\epsilon}{2}$.*

Proof. Given a bilinear group $\mathbb{G}_0$ with prime order p and generator g, a bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_0 \longrightarrow \mathbb{G}_T$ and a random $h \in \mathbb{G}_0$. The DBDH challenger $\mathcal{C}$ randomly selects $a', b', c' \in \mathbb{Z}_p$, $\theta \in \{0, 1\}$, and $\mathcal{R} \in \mathbb{G}_T$. Let $\mathcal{Z} = e(g, g)^{a'b'c'}$, if $\theta = 0$, $\mathcal{R}$ else. Next, $\mathcal{C}$ sends $\mathcal{B}$ the tuple $\langle g, g^{a'}, g^{b'}, g^{c'}, h, h^{b'}, h^{c'}, \mathcal{Z}\rangle$. At last, $\mathcal{B}$ acts as $\mathcal{C}$ in the security game as follows.

- *Initialization*: First of all, $\mathcal{A}$ chooses a challenge access tree $\mathcal{T}^*$ and dispatches it to $\mathcal{B}$.
- *Setup*: In order to generate a public key PK for $\mathcal{A}$, $\mathcal{B}$ needs to select $a', \beta' \in \mathbb{Z}_p$ at random. Next, $\mathcal{B}$ computes $g^\alpha = g^{a'}$, i.e., $\alpha = a'$; $h^\beta = h^{\beta'}(h^{b'})^{a'}$ and $e(g, g)^\beta = e(g, g)^{\beta'} \cdot e(g^{a'}, g^{b'})$, i.e., $\beta = \beta' + a'b'$. $\mathcal{B}$ picks a random s_j for each attribute $a_j \in \mathcal{L}$. If $a_j \in \mathcal{T}^*$, set $PK_j = g^{v_j} = g^{\frac{a'}{s_j}}$, i.e., $v_j = \frac{a'}{s_j}$; otherwise, $PK_j = g^{v_j} = g^{s_j}$, i.e., $v_j = s_j$. Eventually, $\mathcal{B}$ creates $PK = \{g^\alpha, h^\beta, e(g, g)^\beta, \{PK_j \mid \forall a_j \in \mathcal{L}\}\}$ for $\mathcal{A}$.

- *Phase* 1: Here, $\mathcal{A}$ adaptively chooses an attribute set $S \in \mathcal{L}$, and submits it to $\mathcal{B}$ asking for a secret key SK corresponding with S. In response to secret key query from $\mathcal{A}$, $\mathcal{B}$ picks $\hat{r}, \tilde{r} \in \mathbb{Z}_p$ at random, sets $r' = \tilde{r}$ and computes $g^r = \frac{g^{\hat{r}}}{g^{b'}}$, i.e., $r = \hat{r} - b'$. Then, it continues to compute $g^{\beta+\alpha r} = g^{\beta'+a'b'} g^{a'(\hat{r}-b')} = a^{\beta'+a'\hat{r}}$, i.e., $\beta + \alpha r = \beta' + a'\hat{r}$; $g^{\alpha r} h^{r'} = g^{a'\hat{r}} h^{\tilde{r}}$; $g^{r'} = g^{\tilde{r}}$. For each $a_j \in S$, if $a_j \in \mathcal{T}^*$, $\mathcal{B}$ computes $g^{\frac{\alpha r}{v_j}} = g^{\frac{a'(\hat{r}-b')}{a' s_j^{-1}}} = g^{s_j(\hat{r}-b')}$ and $h^{\frac{\alpha r}{v_j}} = h^{s_j(\hat{r}-b')}$; otherwise, $g^{\frac{\alpha r}{v_j}} = g^{\frac{a'(\hat{r}-b')}{s_j^{-1}}}$ and $h^{\frac{\alpha r}{v_j}} = h^{\frac{a'(\hat{r}-b')}{s_j^{-1}}}$. Afterwards, $\mathcal{B}$ answers $\mathcal{A}$ with a secret key $SK = \{\beta + \alpha r, g^{\alpha r} h^{r'}, g^{r'}, \{g^{\frac{\alpha r}{v_j}}, h^{\frac{\alpha r}{v_j}} \mid \forall a_j \in S\}\}$.
- *Challenge*: $\mathcal{A}$ chooses two challenge message m_0, m_1 with a set of keywords $KW^* = \{kw_1^*, kw_2^*, \ldots, kw_t^*\}$ and submits them to $\mathcal{B}$ to be challenged. At first, $\mathcal{B}$ randomly selects $r_1', r_2' \in \mathbb{Z}_p$ to generate $g^{\frac{1}{r_1'}}$ and

$$\begin{aligned} KW_1^* &= \{H(kw_1^*)r_1', H(kw_2^*)r_1', \ldots, H(kw_t^*)r_1'\}; \\ KW_2^* &= \{H(kw_1^*)r_2', H(kw_2^*)r_2', \ldots, H(kw_t^*)r_2'\}. \end{aligned} \tag{38}$$

Then, with the help of fog nodes, $\mathcal{B}$ can construct CT_2^* as

$$CT_2^* = \left\{ g^{\frac{1}{r_1'}}, \{C_1^{i*} = g^{\frac{1}{H(kw_i^*)r_1'}}, C_2^{i*} = g^{\frac{1}{H(kw_i^*)r_2'}} \mid \forall i \in [1,t]\} \right\}. \tag{39}$$

Secondly, $\mathcal{B}$ sends $\mathcal{T}^*$ to fog nodes which chooses a random s_1' and generate $CT_1'^*$ as

$$CT_1'^* = \left\{ \mathcal{T}^*, g^{s_1'}, h^{s_1'}, h^{\beta s_1'}, \{C_j = g^{v_j q_{x^*}(0)} \mid \forall a_j \in \mathcal{T}^*\} \right\}. \tag{40}$$

AT last, $\mathcal{B}$ randomly picks $\theta' \in \{0,1\}$, sets $g^{s_2'} = g^{c'}, h^{s_2'} = h^{c'}$ and computes

$$CT_1^* = \{g^{s_2'}, g^{s_1'} g^{s_2'}, h^{s_1'} h^{s_2'}, CT_1'^*\} \tag{41}$$

and

$$\begin{aligned} C^* &= m_{\theta'} \cdot e(g,g)^{\beta s_2'} \cdot e(g,g)^{\frac{1}{r_2'}} \\ &= m_{\theta'} \cdot \mathcal{Z} \cdot e(g,g)^{\beta' c'} \cdot e(g,g)^{\frac{1}{r_2'}}. \end{aligned} \tag{42}$$

So far, $\mathcal{B}$ can returns $\mathcal{A}$ a complete challenge ciphertext CT^*, where

$$CT^* = \{\mathcal{T}^*, C^* = m_{\theta'} \cdot \mathcal{Z} \cdot e(g,g)^{\beta' c'} \cdot e(g,g)^{\frac{1}{r_2'}}, CT_1^*, CT_2^*\}. \tag{43}$$

- *Phase* 2: This phase is similar to Phase 1.
- *Guess*: $\mathcal{A}$ picks a guess bit θ'' of θ'. If and only if, in the above game, $\theta'' = \theta'$, $\mathcal{B}$ guesses $\theta = 0$ which indicates that $\mathcal{Z} = e(g,g)^{a'b'c'}$. Otherwise, $\mathcal{B}$ guesses $\theta = 1$ i.e., $\mathcal{Z} = \mathcal{R}$.

If $\mathcal{Z} = e(g,g)^{a'b'c'}$, then CT^* is valid, and on this condition, $\mathcal{A}'s$ advantage of guessing θ' is ϵ. Therefore, $\mathcal{B}'s$ probability to guess θ correctly is

$$\Pr\left[\mathcal{B}(g, g^{a'}, g^{b'}, g^{c'}, h, h^{b'}, h^{c'}, \mathcal{Z} = e(g,g)^{a'b'c'}) = 0\right] = \frac{1}{2} + \epsilon. \tag{44}$$

If $\mathcal{Z} = \mathcal{R}$, then CT^* seems completely random to $\mathcal{A}$. Hence,

$$\Pr\left[\mathcal{B}(g, g^{a'}, g^{b'}, g^{c'}, h, h^{b'}, h^{c'}, \mathcal{Z} = \mathcal{R}) = 1\right] = \frac{1}{2}. \tag{45}$$

In conclusion, $\mathcal{B}'s$ advantage to win the above security game is

$$\begin{aligned} Adv(\mathcal{B}) = \frac{1}{2}\Big| &\Pr[\mathcal{B}(g, g^{a'}, g^{b'}, g^{c'}, h, h^{b'}, h^{c'}, \mathcal{Z} = e(g,g)^{a'b'c'}) = 0] \\ &+ \Pr[\mathcal{B}(g, g^{a'}, g^{b'}, g^{c'}, h, h^{b'}, h^{c'}, \mathcal{Z} = \mathcal{R}) = 1]\Big| - \frac{1}{2} = \frac{1}{2}\epsilon. \end{aligned} \tag{46}$$

□

Theorem 2. *Supposed that a PPT adversary $\mathcal{A}$ can break the IND-CKA security of our ABFKS scheme with a non-negligible advantage $\epsilon > 0$, then a PPT simulator $\mathcal{B}$ can be constructed to distinguish a t-DDH tuple from a random tuple with an advantage $\frac{\epsilon}{2}$.*

Proof. For the sake of simplicity, we don't discuss attribute-related issues here, but only the privacy of keywords. Given a bilinear group $\mathbb{G}_0$ with prime order p and generator g. The t-DDH challenger $\mathcal{C}$ randomly selects $x, y_1, y_2, \ldots, y_t \in \mathbb{Z}_p$, $\theta \in \{0,1\}$, and $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_t \in \mathbb{G}_0$. Let $\mathcal{Z}_i = g^{\frac{1}{xy_i}}$ for each $i \in [1,t]$, if $\theta = 0$, $\mathcal{R}_i$ else. Next, $\mathcal{C}$ sends $\mathcal{B}$ the tuple $\langle g, g^{\frac{1}{x}}, \{g^{\frac{1}{y_i}}, \mathcal{Z}_i \mid \forall i \in [1,t]\}\rangle$. Then, $\mathcal{B}$ takes the role of $\mathcal{C}$ in the following security game.

- *Initialization*: At First, $\mathcal{A}$ selects two different challenge keyword sets KW^{0*} and KW^{1*}, each of which contains t keywords in total.
- *Setup*: Since attribute-related issues are not discussed, the public key is not required for $\mathcal{A}$ to generate a ciphertext of keyword.
- *Phase* 1: $\mathcal{A}$ adaptively queries $\mathcal{B}$ for a trapdoor Tk of $KW = \{kw_1, kw_2, \ldots, kw_t\}$ at this phase, where $KW \neq KW^{0*}, KW^{1*}$. In response, $\mathcal{B}$ picks a random $r_3' \in \mathbb{Z}_p$ and computes

$$Tk = (Tk_0, Tk_1) = (g^{r_3'}, \{Tk_1^j = g^{\frac{1}{H(kw_i)r_3'}} \mid \forall i \in [1,t]\}), \tag{47}$$

  then responds $\mathcal{A}$ with Tk.
- *Challenge*: For the challenge keyword sets KW^{0*} and KW^{1*}, $\mathcal{B}$ randomly selects $r_1', r_2' \in \mathbb{Z}_p$, $\theta' \in \{0,1\}$, $H' : \{0,1\}^* \longrightarrow \mathbb{G}_0$, and computes the challenge ciphertext of $KW^{\theta'*} = \{kw_1^{\theta'*}, kw_2^{\theta'*}, \ldots, kw_t^{\theta'*}\}$ as: $g^{\frac{1}{r_1}} = (g^{\frac{1}{x}})^{\frac{1}{r_1'}}$, i.e., $r_1 = xr_1'$; $g^{\frac{1}{H(kw_i^{\theta'*})r_1}} = (g^{\frac{1}{xy_i}})^{\frac{1}{H'(kw_i^{\theta'*})r_1'}} = \mathcal{Z}^{\frac{1}{H'(kw_i^{\theta'*})r_1'}}$, and similarly, $g^{\frac{1}{H(kw_i^{\theta'*})r_2}} =$

$\mathcal{Z}^{\frac{1}{H'(kw_i^{\theta'*})r'_2}}$, i.e., $H(kw_i^{\theta'*}) = y_i H'(kw_i^{\theta'*})$. Then, $\mathcal{B}$ can returns $\mathcal{A}$ a keyword-related challenge ciphertext CT_2^*, where

$$CT_2^* = \left\{ (g^{\frac{1}{xr'_1}}, \{ \mathcal{Z}^{\frac{1}{H'(kw_i^{\theta'*})r'_1}}, \mathcal{Z}^{\frac{1}{H'(kw_i^{\theta'*})r'_2}} \mid \forall i \in [1,t] \} \right\}. \tag{48}$$

- *Phase* 2: This phase is similar to Phase 1.
- *Guess*: $\mathcal{A}$ picks a guess bit θ'' of θ'. If and only if, in the above game, $\theta'' = \theta'$, $\mathcal{B}$ guesses $\theta = 0$ which indicates that $\mathcal{Z}_i = g^{\frac{1}{xy_i}}$ for each $i \in [1,t]$. Otherwise, $\mathcal{B}$ guesses $\theta = 1$ i.e., $\mathcal{Z}_i = \mathcal{R}_i$ for $\forall i \in [1,t]$.

If $\mathcal{Z}_i = g^{\frac{1}{xy_i}}$ for each $i \in [1,t]$, then CT_2^* is available, and under this circumstance, $\mathcal{A}'s$ advantage of guessing θ' is ϵ. Therefore, $\mathcal{B}'s$ probability to guess θ correctly is

$$\Pr\left[\mathcal{B}(g, g^{\frac{1}{x}}, \{g^{\frac{1}{y_i}}, \mathcal{Z}_i = g^{\frac{1}{xy_i}} \mid \forall i \in [1,t]\}) = 0\right] = \frac{1}{2} + \epsilon. \tag{49}$$

If $\mathcal{Z}_i = \mathcal{R}_i$ for $\forall i \in [1,t]$, then CT_2^* seems completely random to $\mathcal{A}$. Hence,

$$\Pr\left[\mathcal{B}(g, g^{\frac{1}{x}}, \{g^{\frac{1}{y_i}}, \mathcal{Z}_i = \mathcal{R}_i \mid \forall i \in [1,t]\}) = 1\right] = \frac{1}{2}. \tag{50}$$

Therefore, $\mathcal{B}'s$ advantage to distinguish a t-DDH tuple from a random tuple is

$$\begin{aligned} Adv(\mathcal{B}) = \frac{1}{2}\Big| &\Pr[\mathcal{B}(g, g^{\frac{1}{x}}, \{g^{\frac{1}{y_i}}, \mathcal{Z}_i = g^{\frac{1}{xy_i}} \mid \forall i \in [1,t]\}) = 0] \\ &+ \Pr[\mathcal{B}(g, g^{\frac{1}{x}}, \{g^{\frac{1}{y_i}}, \mathcal{Z}_i = \mathcal{R}_i \mid \forall i \in [1,t]\}) = 1]\Big| - \frac{1}{2} = \frac{1}{2}\epsilon. \end{aligned} \tag{51}$$

□

Theorem 3. *The ABFKS scheme can achieve peer-peeping resistance.*

Proof. The trapdoor (Ta, Tk) and a ciphertext $CT = \{\mathcal{T}, E_{ck}(M), C = ck \cdot e(g,g)^{\beta s_2} \cdot e(g,g)^{\frac{1}{r_2}}, CT_1, CT_2\}$ maybe eavesdropped by a PPT adversary $\mathcal{A}$ who has the sufficient authority, then $\mathcal{A}$ is able to partially decrypt CT, namely to compute $e(g,g)^{\beta s_2}$. But $\mathcal{A}$ has no idea about the keywords KW embedded in CT nor KW' in Tk, $\mathcal{A}$ can not calculate $e(g,g)^{\frac{1}{r_2}}$. Consequently, even if $\mathcal{A}$ has the same or higher authority as U, i.e., a peer or superior, the CT still cannot be decrypted and peeped by $\mathcal{A}$. □

6.2 Function and Efficiency Comparison

We discuss the function and efficiency issues from a theoretical point of view. Compared with a few up-to-the-minute CP-ABE schemes [16,24,26] in fog computing environment, the ABFKS has richer functions i.e., functional keyword search and peer-peep resistance, as shown in Table 2.

As far as the efficiency is concerned, we compare the computational overhead and storage costs of our scheme with [16] in Table 3 and Table 4 respectively.

Table 2. Functional comparison between previous ABKS schemes and ABFKS.

Schemes	Fine-grained	Keyword search	Attribute update	Access test	Functional keyword search	Peer-peeping resistance
[26]	√		√			
[24]	√	√				
[16]	√	√	√			
ABFKS	√	√	√	√	√	√

Table 3. Comparison of computational overhead between ABFKS and [16].

Algorithm	ABFKS		[16]	
	Fog nodes	User	Fog nodes	User
KeyGen	$SK' : (S+4)g$	$(2S+3)g$	$(2S+4)g$	$(S+1)g$
Enc	$(C_1', C_2') : (3+n+2t)g$	$3g+2e$	$(n+2)g$	$(n+4)g+e$
Trap	$Tk_1 : tg$	$(S+2)g$	$2(S+1)g$	$(2S+1)g$
Search	Access test:$(S+2)e+g$		$(S+1)e+2g$	
	Keyword matching:$\frac{1}{2}(t^2+t+2)e$			
Dec	$(n+3)e$	$e+g$	$(n+2)e$	e

[1] e: Bilinear pairing; g: Exponentiation in group; S: Number of submitted attributes; n: Number of attributes in $\mathcal{T}$; t: Number of keywords.
[2] Search: this algorithm is operated by the CS.
[3] Keyword matching: the possible maximum computational overhead in keyword matching algorithm.

Compared with [16], in order to achieve functional keyword search and peer-peeping resistance, the ABFKS have more computational costs of fog nodes in the encryption and trapdoor generation phases. This is because we need to generate a keyword ciphertext to hide all keywords, so that we can achieve multi-keyword search without complete keyword matching and peer-peeping resistance. For the same reason, the CS has to iterate more times during the search phase. Overall, the computational and storage overhead on the user side has increased very slightly, but the system is more practical and secure.

Table 4. Comparison of storage costs between ABFKS and [16].

Algorithm	ABFKS		[16]	
	Fog nodes	User	Fog nodes	User
KeyGen	$(S+3)\|\mathbb{G}_0\|$	$(2S+2)\|\mathbb{G}_0\|+\|\mathbb{Z}_p\|$	$(2S+3)\|\mathbb{G}_0\|+\|\mathbb{Z}_p\|$	$(S+1)\|\mathbb{G}_0\|+\|\mathbb{Z}_p\|$
Enc	$(3+n+2t)\|\mathbb{G}_0\|$ $+(n+1)\|\mathbb{Z}_p\|$	$(7+n+2t)\|\mathbb{G}_0\|$ $+3\|\mathbb{G}_T\|+(n+4)\|\mathbb{Z}_p\|$	$(n+2)\|\mathbb{G}_0\|$ $+n\|\mathbb{Z}_p\|$	$(n+4)\|\mathbb{G}_0\|$ $+2\|\mathbb{G}_T\|+(n+1)\|\mathbb{Z}_p\|$
Trap	$t\|\mathbb{G}_0\|$	$(S+2)\|\mathbb{G}_0\|+\|\mathbb{Z}_p\|$	$2(S+1)\|\mathbb{G}_0\|+\|\mathbb{G}_T\|+2\|\mathbb{Z}_p\|$	$(2S+1)\|\mathbb{G}_0\|+2\|\mathbb{Z}_p\|$
Search	Access test:$(n+2)\|\mathbb{G}_T\|$		$(S+1)\|\mathbb{G}_T\|$	
	Keyword matching:$\frac{1}{2}(t^2+t+2)\|\mathbb{G}_T\|$			
Dec	$(n+3)\|\mathbb{G}_T\|$	$\|\mathbb{G}_T\|+\|\mathbb{Z}_p\|$	$(n+1)\|\mathbb{G}_T\|$	$\|\mathbb{G}_T\|+\|\mathbb{Z}_p\|$

[1] S: Number of submitted attributes; n: Number of attributes in $\mathcal{T}$; t: Number of keywords; $|G_0|$: Element length in $\mathbb{G}_0$; $|G_T|$: Element length in $\mathbb{G}_T$; $|Z_p|$: Element length in $\mathbb{Z}_p$.

7 ABKFS-PER

For some certain applications, the privacy of access structure and user authority needs to be protected, because it may contain sensitive information. Privacy and efficiency issues haven't been fully considered in the above ABFKS scheme. Therefore, in this section, we provide a construction of ABFKS with privacy preserving, efficient attribute update and reverse outsourcing.

7.1 Construction of ABKFS-PER

Most of the necessary definitions have been given in Section 5. In addition, we define the lagrange-route product. Actually in traditional CP-ABE schemes, the calculation of the lagrange-route product is implied in the decryption process of ciphertexts. It's specifically defined here because it can be outsourced to the cloud to reduce some user's computational costs.

Definition 9 (Lagrange-Route Product). *If $\mathcal{T}'$ is exposed to the CS, the CS can compute the lagrange coefficient of each node in $\mathcal{T}'$. For each leaf node z of $\mathcal{T}'$, there is only one route from it to the root node R. This route passes through some non-leaf nodes, so we can define it by a route set $S_{z\to R} = (x_o, x_1, x_2, \ldots, x_{R-1})$, where $x_0 = z$ and x_{R-1} is R's child node. Then, the lagrange-route product is defined as:*

$$\pi_z = \prod_{x\in S_{z\to R}} \Delta_{i,q_x}(0). \tag{52}$$

The details of ABFKS-PER are shown as follows.

- **Setup**$(1^\lambda, \mathcal{L}) \to (PK, MSK)$: Given a security parameter λ and a set of all possible attributes $\mathcal{L}$, the KAC selects a bilinear group $\mathbb{G}_0$ with prime order p and generator g. Next, it chooses $\alpha, \beta \in \mathbb{Z}_p$ and $h \in \mathbb{G}_0$. For each attribute $a_j \in \mathcal{L}$, it picks out a random v_j and computes $PK_j = g^{v_j}$. Finally, it generates the master key MSK and publishes the public key PK.

$$PK = \left\{\mathbb{G}_0, g, h, g^\alpha, e(g,g)^\beta, \{PK_j = g^{v_j} \mid \forall a_j \in \mathcal{L}\}\right\}; \tag{53}$$

$$MSK = \{\alpha, \beta, \{v_j \mid \forall a_j \in \mathcal{L}\}\}. \tag{54}$$

- **KeyGen**$(MSK, S, U_{id}) \to (SK, SK')$: While receiving an attribute set S and U_{id} from the U, the KAC randomly selects $r, r' \in \mathbb{Z}_p$ and $r_i \in \mathbb{Z}_p$ for each $a_i \in \mathcal{L} \setminus S$, and computes $d = Hash(U_{id} \parallel r \parallel r')$, then generates the user secret key SK and the randomized attribute secret key with error SK' and the auxiliary decryption secret key SK'' as follows.

$$SK = \{d\}; \tag{55}$$

$$SK' = \left\{\{SK'_j = g^{\frac{\alpha r}{v_j d}} \mid \forall a_j \in S\}, \{SK'_i = g^{r_i} \mid \forall a_i \in \mathcal{L} \setminus S\}\right\} \tag{56}$$

$$SK'' = \{\frac{\beta + \alpha r}{d}, g^{\frac{\alpha r}{d}} h^{\frac{r'}{d}}, g^{\frac{r'}{d}}\}. \tag{57}$$

Specifically, the SK' is composed of real randomized attribute secret keys $\{g^{\frac{\alpha r}{v_j d}} \mid \forall a_j \in S\}$ and fake keys $\{g^{r_i} \mid \forall a_i \in \mathcal{L} \setminus S\}$. The KAC reorders each SK' in accordance with attribute sequence number, i.e., the index of a_i where $a_i \in S$, and sends SK and SK'' to the U, sends (U_{id}, SK') to the CS. Afterwards, the CS can construct Table 5 to store U_{id} and SK'.

Table 5. User's attribute secret keys stored in the cloud.

User ID	Attribute secret key
U_{id}	SK'
$\vdots$	$\vdots$

In the view of the CS, each user has all attribute secret keys, and it can not distinguish real randomized attribute secret keys with fake keys. Therefore, the privacy of user authority is preserved against the CS.

- **Enc**$(PK, \mathcal{T}, M, KW) \to CT$: The DO encrypts the symmetric encryption key ck as follows.
  1. **Attribute Ciphertext**: The DO chooses an access tree $\mathcal{T}$, which describes an access policy. Let $\mathcal{X}$ be a set of attributes corresponding with all leaf nodes in $\mathcal{T}$. Assume there are totally n leaf nodes in $\mathcal{T}$, where $2 \mid \mathcal{X} \mid \leq L$. Each leaf node stands for an attribute i.e., $a_i = att(z_i) \in \mathcal{X}$. On the basis of $\mathcal{T}$, the DO construct a new access tree $\mathcal{T}'$ as follows. For each $z_i \in \mathcal{T}$, the DO replace z_i by an OR gate node which is named $z_i - or$ node. The $z_i - or$ node has two child nodes i.e., real leaf node z_i and fake leaf node z_i', where $att(z_i') \notin \mathcal{X}$ and $att(z_i') \neq att(z_j')$ for $i \neq j$. The DO randomly chooses one from z_i and z_i' as the left child of $z_i - or$ node. If the left child of $z_i - or$ node is z_i, then the DO sets a bit $rn_i = 0$, otherwise $rn_i = 1$. Eventually, the DO can construct a new access tree $\mathcal{T}'$ as well as a real-leaf-node bit string $RN = rn_1 \parallel rn_2 \parallel \ldots \parallel rn_n$.

     The DO sends an access tree $\mathcal{T}'$ to fog nodes, which describes an access policy. Fog nodes randomly chooses a polynomial q_x for each node x of $\mathcal{T}'$ from the root node R in a top-down manner. For each node x of $\mathcal{T}'$, $d_x = k_x - 1$, where d_x is the degree of q_x and k_x is the threshold value of x. Beginning with root node R, fog nodes randomly pick $s_1, s_1' \in \mathbb{Z}_p$ and set $q_R^{ck}(0) = s_1, q_R^{RN}(0) = s_1'$. Next, they randomly choose d_R other points of q_R to define the polynomial completely. For any other node x, fog nodes set $q_x(0) = q_{parent(x)}(index(x))$ and choose d_x other point to define q_x completely. Let $\mathcal{X}'$ be a set of attributes corresponding with all leaf nodes in $\mathcal{T}'$. In this way, fog nodes construct CT'_{ck} and CT'_{RN} respectively, and send them back to the DO.

$$CT'_{ck} = \left\{\mathcal{T}', g^{s_1}, h^{s_1}, \{C_j^{ck} = g^{v_j q_x(0)} \mid \forall a_j = att(x) \in \mathcal{X}'\}\right\}; \quad (58)$$

$$CT'_{RN} = \left\{\mathcal{T}', g^{s_1'}, h^{s_1'}, \{C_j^{RN} = g^{v_j q_y(0)} \mid \forall a_j = att(y) \in \mathcal{X}'\}\right\}. \quad (59)$$

2. **Keyword Ciphertext**: The process of generating keyword ciphertext is the same as ABFKS. So, fog nodes compute CT'_{KW} and send it back to the DO.

$$CT'_{KW} = \{C^i_{KW_1} = g^{\frac{1}{H(kw_i)r_1}}, C^i_{KW_2} = g^{\frac{1}{H(kw_i)r_2}} \mid \forall i \in [1, t]\}. \tag{60}$$

3. For each $att(z_j) \in \mathcal{X}' \setminus \mathcal{X}$, the DO randomly chooses $e_j \in \mathbb{Z}_p$, and generates a ciphertext with error $\widetilde{CT'_{ck}}$ by CT'_{ck} as

$$\widetilde{CT'_{ck}} = \left\{ \begin{array}{l} \mathcal{T}', g^{s_1}, h^{s_1}, \{C^{ck}_i = g^{v_i q_x(0)} \mid \forall a_i = att(x) \in \mathcal{X}\}, \\ \{C^{ck}_j = g^{e_j} \mid \forall a_j = att(z) \in \mathcal{X}' \setminus \mathcal{X}\} \end{array} \right\}. \tag{61}$$

For $i \in [1, 2n]$, the DO reorders each attribute ciphertext CT^{ck}_i in $\widetilde{CT'_{ck}}$ according to its attribute sequence number, i.e. the index of $att(z_i)$, where $att(z_i) \in \mathcal{X}'$. The DO picks $s_2, s'_2 \in \mathbb{Z}_p$ at random and generates CT_{ck}, CT_{RN}, CT_{KW} as $CT_{ck} = \{g^{s_2}, g^{s_1} g^{s_2}, h^{s_1} h^{s_2}\}$, $CT_{RN} = \{g^{s'_2}, g^{s'_1} g^{s'_2}, h^{s'_1} h^{s'_2}\}$, $CT_{KW} = \{g^{\frac{1}{r_1}}, CT'_{KW}\}$. Then, it computes $e(g,g)^{\beta s_2}$, $e(g,g)^{\beta s'_2}$ and $e(g,g)^{\frac{1}{r_2}}$ to generate a pair of final ciphertexts: CT_U and CT_{NU} as follows.

$$CT_U = \left\{ \begin{array}{l} \mathcal{T}', E_{ck}(M), C_{ck} = ck \cdot e(g,g)^{\beta s_2} \cdot e(g,g)^{\frac{1}{r_2}}, \\ C_{RN} = RN \cdot e(g,g)^{\beta s'_2} \cdot e(g,g)^{\frac{1}{r_2}}, CT_{ck}, CT_{RN}, CT_{KW} \end{array} \right\}; \tag{62}$$

$$CT_{NU} = \{\widetilde{CT'_{ck}}, CT'_{RN}\}. \tag{63}$$

Both CT_U and CT_{NU} are stored in the CS, and only CT_U can be accessed by users. CT_{NU} consists of many attribute ciphertexts, which are not allowed for user to access.

- **Trap**$(KW') \to Tk$: From the CS point of view, every user has all the attributes, so there is no access test process and the U doesn't need to generate the attribute trapdoor Ta by his secret key while searching for a ciphertext. Moreover, the attribute secret keys of U, i.e., SK', are randomized and stored in the cloud, so the U can not generate the Ta locally as in the ABFKS, so that the U only needs to generate the keyword trapdoor Tk . The generation process is the same as ABFKS, so

$$Tk = (Tk_0, Tk_1) = (g^{r_3}, \{Tk^j_1 = g^{H(kw'_j)r_3} \mid \forall j \in [1, t]\}). \tag{64}$$

- **Search**$(CT_U, Tk) \to \perp$ *or* $l \in [1, t]$: From the CS point of view, the U has all the attributes, so the access test process can be skipped. If there is at least one keywords to be identical in KW and KW', the **Search** algorithm outputs $l \in [1, t]$, which means there are l keywords in common between KW and KW'. Otherwise, the algorithm outputs $\perp$.

  Keyword Matching: This process is the same as ABFKS, so the CS ultimately generates the following Table 6 and sorts it in descending order according to the correlation, supposed that there are N ciphertexts such that $l \neq 0$ in total.

- **PreDec**$(CT_{NU}, SK') \to (F^{ck}, F^{RN})$: Before downloading ciphertexts, the CS can do some pre-calculation in order to reduce the computation overhead for user's decryption. This algorithm consists of two sub-algorithms: **PreDec**$_{ck}$ and **PreDec**$_{RN}$.

Table 6. The result generated by the search algorithm.

Ciphertext	Matching result vector	Correlation
$CT_{U(1)}$	$\overrightarrow{mr}_{(1)}$	$l_{(1)}$
$CT_{U(2)}$	$\overrightarrow{mr}_{(2)}$	$l_{(2)}$
$\vdots$	$\vdots$	$\vdots$
$CT_{U(N)}$	$\overrightarrow{mr}_{(N)}$	$l_{(N)}$

1. $\mathbf{PreDec}_{ck}(\mathcal{T}', SK', \widetilde{CT'_{ck}}) \to \overrightarrow{F_z^{ck}}$: For each leaf node x of $\mathcal{T}'$, supposed $a_i = att(x)$. The CS computes $F_x^{ck} = F_x^{ck}(C_i^{ck}, SK_i', x)$ as:

$$F_x^{ck}(C_i^{ck}, SK_i', x) = e(g^{v_i q_x(0)}, g^{\frac{\alpha r}{v_i d}}) = e(g,g)^{\frac{\alpha r q_x(0)}{d}}. \tag{65}$$

If and only if C_i^{ck} is a real attribute ciphertext and SK_i' is a real randomized attribute secret key, the above equation holds. Otherwise, F_x^{ck} is random in $\mathbb{G}_T$. Then the CS can construct a precomputation vector $\overrightarrow{F_{ck}}$, where

$$\overrightarrow{F^{ck}} = \{(F_{x_1}^{ck})^{\pi_{x_1}}, (F_{x_2}^{ck})^{\pi_{x_2}}, \ldots, (F_{x_{2n}}^{ck})^{\pi_{x_{2n}}}\}. \tag{66}$$

2. $\mathbf{PreDec}_{RN}(\mathcal{T}', SK', CT'_{RN}) \to \overrightarrow{F_y^{RN}}$: For each leaf node y of $\mathcal{T}'$, supposed $a_j = att(y)$. The CS computes $F_y^{RN} = F_y^{RN}(C_j^{RN}, SK_j', y)$ as:

$$F_y^{RN}(C_j^{RN}, SK_j', y) = e(g^{v_j q_y(0)}, g^{\frac{\alpha r}{v_j d}}) = e(g,g)^{\frac{\alpha r q_y(0)}{d}}. \tag{67}$$

Since every C_j^{RN} is real, the above equation holds only when SK_j' is a real randomized attribute secret key. Otherwise, F_y^{RN} is random in $\mathbb{G}_T$. Then the CS can construct a precomputation vector $\overrightarrow{F_{RN}}$, where

$$\overrightarrow{F^{RN}} = \{(F_{y_1}^{RN})^{\pi_{y_1}}, (F_{y_2}^{RN})^{\pi_{y_2}}, \ldots, (F_{y_{2n}}^{RN})^{\pi_{y_{2n}}}\}. \tag{68}$$

Finally the CS constructs Table 7 for U to access as follows.

Table 7. Access table for the user generated by the cloud.

Ciphertext	Matching result vector	Correlation	Precomputation 1	Precomputation 2
$CT_{U(1)}$	$\overrightarrow{mr}_{(1)}$	$l_{(1)}$	$\overrightarrow{F^{ch}}_{(1)}$	$\overrightarrow{F^{RN}}_{(1)}$
$CT_{U(2)}$	$\overrightarrow{mr}_{(2)}$	$l_{(2)}$	$\overrightarrow{F^{ck}}_{(2)}$	$\overrightarrow{F^{RN}}_{(2)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$CT_{U(N)}$	$\overrightarrow{mr}_{(N)}$	$l_{(N)}$	$\overrightarrow{F^{ck}}_{(N)}$	$\overrightarrow{F^{RN}}_{(N)}$

- **Dec**$(CT_U, SK, SK'', \overrightarrow{F^{ck}}, \overrightarrow{F^{RN}}) \to ck$: The U is allowed to download the ciphertext CT in Table 7 according to l, and he can decrypt it with the help of $\overrightarrow{F^{ck}}$ and $\overrightarrow{F^{RN}}$, if and only if his attribute set $S \models \mathcal{T}$. The U checks whether his attribute set $S \models \mathcal{T}'$ or not. If $S \not\models \mathcal{T}'$, the U cannot decrypt CT. Otherwise, the U conducts the following operations.
  1. Since $S \models \mathcal{T}'$, for each leaf node in $\mathcal{T}'$, the U is enabled to pick out $F_{y_i}^{RN}$ from $\overrightarrow{F^{RN}}$, where $att(y_i) \in S \bigcap \mathcal{X}'$. Then the U computes P_{RN} as
  $$P_{RN} = \prod_{att(y) \in S \bigcap \mathcal{X}'} (F_y^{RN})^{\pi_y} = e(g,g)^{\frac{\alpha r s_1'}{d}}. \quad (69)$$
  Then the U sends the auxiliary decryption secret key SK'' to fog nodes. The fog nodes interact SK'' and CT_{RN} to compute Q_{RN} as
  $$Q_{RN} = e(g,g)^{\frac{\beta s_2'}{d} - \frac{\alpha r s_1'}{d}} = \frac{e(g^{\frac{\beta+\alpha r}{d}}, g^{s_2'}) \cdot e(g^{\frac{r'}{d}}, h^{s_1'} h^{s_2'})}{e(g^{\frac{\alpha r}{d}} h^{\frac{r'}{d}}, g^{s_1'} g^{s_2'})}, \quad (70)$$
  and send Q_{RN} back to the U. Then, the U computes PQ_{RN} as $PQ_{RN} = (P_{RN} \cdot Q_{RN})^d = e(g,g)^{\beta s_2'}$ and obtains $\overrightarrow{mr}$ in Table 7 related to CT. There is at least one mr_j in $\overrightarrow{mr}$ such that $mr_j = i \neq 0$, i.e., $kw_j' = kw_i$. Thus, the U can make use of the j^{th} keyword in KW' and i^{th} keyword in KW to compute $E = e(g,g)^{\frac{1}{r_2}}$. Finally, the U calculates RN as
  $$RN = \frac{C_{RN}}{PQ_{RN} \cdot E}. \quad (71)$$
  2. So far, the U can reveal $\mathcal{T}$ from $\mathcal{T}'$, and check whether his attribute set $S \models \mathcal{T}$. If $S \not\models \mathcal{T}$, the U cannot decrypt CT. Otherwise, the U continues to decrypt the ciphertext. Obtaining RN, the U is enable to pick out $F_{x_i}^{ck}$ from $\overrightarrow{F^{ck}}$, where $att(x_i) \in S \bigcap \mathcal{X}$. Then the U computes P_{ck} as
  $$P_{ck} = \prod_{att(x) \in S \bigcap \mathcal{X}} (F_x^{ck})^{\pi_x} = e(g,g)^{\frac{\alpha r s_1}{d}}. \quad (72)$$
  The fog nodes interact SK'' and CT_{ck} to compute Q_{ck} as
  $$Q_{ck} = e(g,g)^{\frac{\beta s_2}{d} - \frac{\alpha r s_1}{d}} = \frac{e(g^{\frac{\beta+\alpha r}{d}}, g^{s_2}) \cdot e(g^{\frac{r'}{d}}, h^{s_1} h^{s_2})}{e(g^{\frac{\alpha r}{d}} h^{\frac{r'}{d}}, g^{s_1} g^{s_2})} \quad (73)$$
  and send Q_{ck} back to the U. Then, the U computes $PQ_{ck} = (P_{ck} \cdot Q_{ck})^d = e(g,g)^{\beta s_2}$ and finally calculates ck as
  $$ck = \frac{C_{ck}}{PQ_{ck} \cdot E}. \quad (74)$$

  So far, the U can decrypt $E_{ck}(M)$ with ck by symmetric decryption.
- **Attribute update**: When the attribute set of U changes from S to S_{new}, the KAC only needs to run **KeyGen**(Msk, S_{new}, U_{id}) to generate a new secret key SK_{new}, SK'_{new} and SK''_{new} to replace the original ones.

7.2 Reverse Outsourcing

It is worth noting that the above two sub-algorithms **PreDec**$_{ck}$ and **PreDec**$_{RN}$ can be reversely outsourced to idle users with intelligent devices, thus reducing the computational burden of cloud servers. Hence, we define the concept of reverse outsourcing.

Definition 10 (Reverse Outsourcing). *As is known to all, the cloud service provider can provide outsourcing services for end users to reduce their local computational burden. However, the reverse outsourcing is on the contrary. There are innumerable users all over the world, whose intelligent devices are idle and connected to the Interne. We can call them "idle users" and each of them can provide a small amount of computational resource for the cloud. In order to reduce the cloud computational overhead, the cloud can divide a computational task into several parts and outsource them to different idle users respectively. It must be noted that, the reverse outsourcing has to prevent sensitive information leakage.*

When the CS outsources a computational task to idle users, they must follow the protocol specification. If the reverse outsourcing computational task is checked valid, the corresponding idle users can be rewarded by the CS. In this paper, the reverse outsourcing is applied to rational idle user model, which is defined as follows .

Definition 11 (Rational Idle User Model). *Rational idle user are selfish and lazy, and always attempt to maximize their profits, which means that they prefer to get rewards from the CS, rather than save the computational resource of their idle smart devices. Therefore, for each rational idle user U_i, it holds that $ut_i^{++} > ut_i^{+} > ut_i^{-} > ut_i^{--}$, where*
- *ut_i^{++} is the utility of U_i when he can get rewards without following the protocol specification.*
- *ut_i^{+} is the utility of U_i when he follows the protocol specification and gets rewards.*
- *ut_i^{-} is the utility of U_i when he doesn't get rewards without following the protocol specification.*
- *ut_i^{--} is the utility of U_i when he follows the protocol specification but doesn't get rewards.*

In rational idle user model, any system user is independent from each other. Since the performance of each user U_i satisfies $ut_i^{++} > ut_i^{+} > ut_i^{-} > ut_i^{--}$, this means that U_i have different strategies. In order to analyze the best strategy for a rational idle user, we formalize the reverse outsourcing game by means of game theory and introduce the notion of Nash equilibrium.

Definition 12 (Reverse Outsourcing Game). *The reverse outsourcing game is a tuple $G_{RO} = \{U, T, ST, R, V\}$, where*

- $U = \{U_1, U_2, \ldots, U_n\}$ *is the set of* n *rational idle users, where* $n \geq 1$*. Each of them needs to complete a computational task in order to get rewards from the* CS*.*
- $T = \{T_1, T_2, \ldots, T_n\}$ *is the set of computational tasks, where* T_i *is assigned to* U_i*.*
- $ST = \{ST_1, ST_2, \ldots, ST_n\}$ *is the set of rational idle users' strategies in* G_{RO}*. In particular,* $ST_i = \{st_i^0, st_i^1\} \in ST$ *is the set of* U_i*'s strategies.* st_i^0 *denotes that* U_i *wants to be rewarded without following the protocol specification;* st_i^1 *denotes that* U_i *follows the protocol honestly.*
- $R = \{R_1, R_2, \ldots, R_n\}$ *is the set of computational results , where* R_i *is the result of* T_i*.*
- V *is a verification algorithm to check whether* R *is valid or not. If* R *is valid, i.e., each* R_i *is valid, then every rational idle user will get the same reward. Otherwise, none of them will get anything.*

Definition 13 (Nash Equilibrium of G_{RO}). *For a given strategy* $ST^* = (st_1^*, st_2^*, \ldots, st_n^*)$*,* ST^* *is Nash equilibrium for* G_{RO}*, if and only if for any rational idle user* $U_i \in U$*, when the game* G_{RO} *is finished, for any* $st_i \in ST_i$*, it holds that*

$$ut_i(st_i^*, st_j^*) \geq ut_i(st_i, st_j^*), \tag{75}$$

where $st_i^* \in ST_i$*.*

In our scheme, either $PreDEC_{ck}$ or $PreDec_{RN}$ can be reversely outsourced to a set of rational idle users U. For instance, for each leaf node x of $\mathcal{T}'$, the CS sends a tuple $\{C_i^{ck}, SK_i', x\}$ to a rational idle user U_i and asks him to compute the function $F_x^{ck} = e(C_i^{ck}, SK_i')$. Under the discrete logarithm assumption, the probability that U_i does not follow the protocol but obtains the correct result $e(g,g)^{\frac{\alpha r q_x(0)}{d}}$ is negligible. If U_i cheats, for example, by randomly generating an incorrect result, the cheating behavior can be detected with only a small improvement on the original scheme. While generating ciphertext, the DO adds $Hash(RN)$ and $Hash(ck)$ into CT_U. When an end user decrypts CT_U, if his attribute set $S \models \mathcal{T}'$ or $S \models \mathcal{T}$, but he can't calculate the correct RN or ck, he can report an error to the CS. Then, every participant U_i will not be rewarded by the CS. According to Nash equilibrium theory, a rational idle user doesn't follow the protocol to generate wrong computational results, which will not increase his utility but consume his computational resources without rewards. Therefore, if and only if all rational idle users implement the protocol honestly, the profit of each user can be maximized.

7.3 Analysis of ABKFS-PER

The security proof of this scheme is similar to that of ABFKS, so we omit it here. The ABFKS-PER also achieves IND-CPA and IND-CKA security as well as peer-peeping resistance. Since many ABKS schemes, including ABFKS, fail to protect the privacy of access structure and user authority and do not support efficient attribute update, we have proposed the ABFKS-PER, which supports privacy preserving, efficient attribute update and reverse outsourcing. Specifically, the privacy of access structure is protected by replacing each leaf node with an $z_i - or$ node. The $z_i - or$ node has two child nodes i.e., real leaf node z_i and fake leaf node z_i'. Supposed that there are n leaf nodes in $\mathcal{T}$, the probability of the cloud to recover $\mathcal{T}$ from $\mathcal{T}'$ is 2^{-n}. For the user, only SK and SK'' can be obtained, all attribute secret keys (SK') are stored in the cloud. By filling up fake attribute secret keys, every user has all the attributes in the view of cloud so that the privacy of user authority is also protected. When a user requests to update an attribute, the KAC only needs to generate a set of new keys (SK, SK', SK'') for this user, instead of updating everyone's corresponding attribute secret key as previous schemes [16, 26]. Considered that there are countless idle intelligent devices connected to the Internet all over the world, which can provide computing resources for the cloud, we initially propose the concept of reverse outsourcing. In the rational idle user model,the cloud is allowed to outsource computational tasks to rational idle users. As far as we are concerned, we think reverse outsourcing may be a new trend in cloud computing.

8 Conclusion

In this paper, we propose an attribute-based encryption with functional keyword search (ABKFS) scheme in fog computing environment at first. The ABFKS initially achieves functional keyword search and peer-peeping resistance, which makes it more practical and secure. The strict security proof has shown that it is selective CPA and CKA security. Then, we present a construction of ABFKS with privacy preserving, efficient attribute update and reverse outsourcing (ABFKS-PER). In this scheme, the privacy of access structure and user authority are both preserved against the cloud and efficient attribute update makes the scheme more practical. For completeness, we propose a novel concept of reverse outsourcing, which maybe a new trend in cloud computing. In the future, we will not only continue to focus on issues of cloud computing, fog computing and edge computing, but also consider data mining and machine learning based on encrypted data.

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: 2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA. pp. 321–334 (2007)
2. Boneh, D., Crescenzo, G.D., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings. pp. 506–522 (2004)
3. Bonomi, F., Milito, R.A., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCC@SIGCOMM 2012, Helsinki, Finland, August 17, 2012. pp. 13–16 (2012)

4. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China. pp. 829–837 (2011)
5. Cheung, L., Newport, C.C.: Provably secure ciphertext policy ABE. In: Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007. pp. 456–465 (2007)
6. Cui, H., Deng, R.H., Liu, J.K., Li, Y.: Attribute-based encryption with expressive and authorized keyword search. In: Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part I. pp. 106–126 (2017)
7. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006. pp. 89–98 (2006)
8. Green, M., Hohenberger, S., Waters, B.: Outsourcing the decryption of ABE ciphertexts. In: 20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings (2011)
9. Green, M., Hohenberger, S., Waters, B.: Outsourcing the decryption of ABE ciphertexts. In: 20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings (2011)
10. Horváth, M.: Attribute-based encryption optimized for cloud computing. In: SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Sněžkou, Czech Republic, January 24-29, 2015. Proceedings. pp. 566–577 (2015)
11. Li, H., Yang, Y., Luan, T.H., Liang, X., Zhou, L., Shen, X.S.: Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data. IEEE Trans. Dependable Sec. Comput. **13**(3), 312–325 (2016)
12. Li, J., Shi, Y., Zhang, Y.: Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage. International Journal of Communication Systems **30**(1), n/a–n/a (2015)
13. Li, J., Huang, X., Li, J., Chen, X., Xiang, Y.: Securely outsourcing attribute-based encryption with checkability. IEEE Trans. Parallel Distrib. Syst. **25**(8), 2201–2210 (2014)
14. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA. pp. 441–445 (2010)
15. Miao, Y., Ma, J., Liu, X., Wei, F., Liu, Z., Wang, X.A.: m2-abks: Attribute-based multi-keyword search over encrypted personal health records in multi-owner setting. J. Medical Systems **40**(11), 246:1–246:12 (2016)
16. Miao, Y., Ma, J., Liu, X., Weng, J., Li, H., Li, H.: Lightweight fine-grained search over encrypted data in fog computing. IEEE Transactions on Services Computing pp. 1–1 (2018)
17. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. pp. 457–473 (2005)

18. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000. pp. 44–55 (2000)
19. Stojmenovic, I., Sheng, W.: The fog computing paradigm: Scenarios and security issues. In: Federated Conference on Computer Science and Information Systems, IEEE Computer Society. pp. 1–8 (2014)
20. Stojmenovic, I., Wen, S., Huang, X., Luan, H.: An overview of fog computing and its security issues. Concurrency and Computation: Practice and Experience **28**(10), 2991–3005 (2016)
21. Sun, W., Yu, S., Lou, W., Hou, Y.T., Li, H.: Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. IEEE Trans. Parallel Distrib. Syst. **27**(4), 1187–1198 (2016)
22. Wang, H., Dong, X., Cao, Z., Li, D.: Secure and efficient attribute-based encryption with keyword search. Comput. J. **61**(8), 1133–1142 (2018)
23. Wang, S., Zhou, J., Liu, J.K., Yu, J., Chen, J., Xie, W.: An efficient file hierarchy attribute-based encryption scheme in cloud computing. IEEE Trans. Information Forensics and Security **11**(6), 1265–1277 (2016)
24. Xiao, M., Zhou, J., Liu, X., Jiang, M.: A hybrid scheme for fine-grained search and access authorization in fog computing environment. Sensors **17**(6), 1423 (2017)
25. Yi, S., Li, C., Li, Q.: A survey of fog computing: Concepts, applications and issues. In: Proceedings of the 2015 Workshop on Mobile Big Data, Mobidata@MobiHoc 2015, Hangzhou, China, June 21, 2015. pp. 37–42 (2015)
26. Zhang, P., Chen, Z., Liu, J.K., Liang, K., Liu, H.: An efficient access control scheme with outsourcing capability and attribute update for fog computing. Future Generation Comp. Syst. **78**, 753–762 (2018)
27. Zhang, R., Ma, H., Lu, Y.: Fine-grained access control system based on fully outsourced attribute-based encryption. Journal of Systems and Software **125**, 344–353 (2017)
28. Zheng, Q., Xu, S., Ateniese, G.: VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In: 2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014. pp. 522–530 (2014)
29. Zhu, H., Mei, Z., Wu, B., Li, H., Cui, Z.: Fuzzy keyword search and access control over ciphertexts in cloud computing. In: Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part I. pp. 248–265 (2017)
30. Zuo, C., Shao, J., Wei, G., Xie, M., Ji, M.: Cca-secure ABE with outsourced decryption for fog computing. Future Generation Comp. Syst. **78**, 730–738 (2018)