Towards Privacy-Preserving and Efficient Attribute-Based Multi-Keyword Search

Zhidan Li, Wenmin Li, Fei Gao, Wei Yin, Hua Zhang, Qiaoyan Wen, Kaitai Liang

Abstract—Searchable encryption can provide secure search over encrypted cloud-based data without infringing data confidentiality and data searcher privacy. In this work, we focus on a secure search service providing fine-grained and expressive search functionality, which can be seen as a general extension of searchable encryption and called attribute-based multi-keyword search (ABMKS). In most of the existing ABMKS schemes, the ciphertext size of keyword index (encrypted index) grows linearly with the number of the keyword associated with a file, so that the computation and communication complexity of keyword index is limited to \(O(m)\), where \(m\) is the number of the keyword. To address this shortage, we propose the first ABMKS scheme through utilizing keyword dictionary tree and the subset cover, in such a way that the ciphertext size of keyword index is not dependent on the number of underlying keyword in a file. In our design, the complexity of computation and the complexity of the keyword index are at most \(O(2\log(n/2))\) for the worst case, but \(O(1)\) for the best case, where \(n\) is the number of keyword in a keyword dictionary. We also present the security and the performance analysis to demonstrate that our scheme is both secure and efficient in practice.

Index Terms—searchable encryption, keyword dictionary tree, subset cover, attribute-based multi-keyword search, encrypted index.

1 Introduction

Being offered advanced cloud-based data storage [1], [2] and the cloud computing services [3], Internet data owners (DO) prefer to outsource the massive amount of data and expensive data computation to remote cloud. To rapidly occupy a piece of data outsourcing market, cloud service providers (CSP), such as Alibaba Cloud, Google AppEngine, and Dropbox, have prepared themselves to provide flexible and multi-functional data services for Internet data users (DU) to meet various needs, e.g., data sharing, retrieving, and cooperative computing [4], [5], [6]. However, data privacy and access control over the outsourced data must be guaranteed in practice. To this end, DO may be recommended to encrypt their data via some encryption techniques (such as AES, RSA) before outsourcing the data to CSP. This approach can protect the data privacy, security and somewhat data integrity. Nevertheless, it also brings side effects, at the same time, to DU and CSP. For example, if the outsourced data is encrypted, standard plaintext-based search mechanism may be applicable no more. This paper deals with the case where a privacy-preserving search over encrypted data is needed.

In order to address the search problem over encrypted data, searchable encryption (SE) has been proposed in the literature. Combining encryption with secure search, it can be used to protect privacy of search query but also the confidentiality of outsourced data. Specifically, it enables a valid data user to generate a trapdoor based on some specified query “description” for cloud server, so that the server can use the trapdoor to search over the encrypted data without violating data confidentiality and search privacy.

Song et al. [7] seminally proposed the notion of keyword search over encrypted data, called Searchable Symmetric Encryption (SSE), in which users have to securely share key for data encryption. Later, SE was designed in the public-key context [13], [14], [15], which can resolve the problem of secret key distribution yielded by SSE. The follow-up research works [19], [20], [21], [22], [23], [25], [26], [27] were proposed to achieve various functionalities under the umbrella of public key based SE, for example, these works [19], [20], [21] focus on single keyword query, while the papers [25], [27] consider to enable users to make multi-keyword search.

In practice, DOs may prefer to maintain secure access control over their outsourced data, so that the unauthorized data user can not gain access to the data. One of the promising technologies called ciphertext-policy attribute-based encryption (CP-ABE) has been used to inject access control policy into data encryption. Zheng et al. [19] proposed a new primitive called attribute-based keyword search (CP-ABKS) by integrating SE and ABE. In their CP-ABKS schemes, an access policy tree is associated with ciphertext and a private key is associated with user attributes. If the attributes satisfy the policy tree and the trapdoor matches the keywords ciphertext (encrypted indexes) simultaneously, then the ciphertext can be located and further decrypted. Some follow-up research works [26], [27] that are proposed to achieve multi-keyword search based on [19]. However, there is a limitation in most of the existing attribute-based multiple keyword search schemes (ABMKS), e.g., [26], [27]. Due to the number of keyword increases in the multi-keyword search, the size of encrypted index and the computational complexity are bounded by \(O(m)\), where \(m\) is the number of keyword embedded in a file.

In this paper, inspired by the CP-ABKS schemes, we de-
sign a Towards Privacy-Preserving and Efficient Attribute-Based Multi-Key-Keyword Search (TPPE-ABMKS) through utilizing keyword dictionary tree and the subset cover. Our TPPE-ABMKS can achieve multi-keyword search with fine-grained access control and the number of the encrypted index is relatively small, which is not dependent on the number of underlying keywords in a file. The contribution of this paper is summarized as follows.

1) We, for the first time, provide a secure multi-keyword search service with short-size ciphertext of keyword index. Our number of the encrypted index is not linear with the number of keyword embedded in a file. To the best of our knowledge, our design is the first of its type that achieves short-size ciphertext of keyword index in the model of MKS. More specifically, by designing the tools of keyword dictionary tree and the subset cover, in our scheme, the complexity of computation and the complexity of the encrypted index are at most $O(2 \log (n/2))$ for the worst case but $O(1)$ for the best case, where $n$ is the number of keyword in a keyword dictionary in system.

2) We show that our design is selectively secure against the chosen-keyword attack via the formal security analysis, and our performance evaluation proves that the scheme is efficient in terms of both the computation and communication overhead, in particular, the time cost of the ciphertext generation and data retrieve are more efficient than that of the existing ABMKS schemes.

The rest of this paper is organized as follows. In Section 2, we present an overview of related work. In Section 3, we briefly review some basic primitives which will be used in this paper and also define the main building blocks for our construction. We then present the system and threat models of our scheme, the construction, and the security model in Section 4. In Section 5, the proposed scheme is presented. Section 6 presents the security analysis of the proposed scheme. Section 7 shows the experimental analysis of the proposed scheme and the comparison with some related works. We conclude the paper in Section 8.

## 2 Related Work

SE was first proposed by Song et al. [7], which allows DOs or DUs to generate a trapdoor that can be used by cloud server to search over the outsourced encrypted data. The existing SE schemes can be categorized into two branches: SEs in the symmetric-key context [7], [9], [10], [11], [12] and the ones in public-key scenario [13], [14], [15], [16], [17], [18]. In order to quickly obtain the encrypted record, researchers have been working on the keyword-based information retrieval to achieve various functionalities and practical needs. The research works in [9], [11] have been proposed SE in the multi-users setting, where DOs can enforce an access control policy by distributing some secret keys to authorized system. Some SE schemes focus on improving the level of keyword security and privacy. Chen et al. [23] proposed a public key encryption with keyword search in dual-server model, which can resist the keyword guessing attack. Later on, Liu et al. [6] and Miao et al. [27] presented the verifiable SE schemes by using public-auditing technique, respectively. Their schemes guarantee that DUs can prevent the semi-trusted CSP from tampering the sharing data and returning false search results.

Attribute-based encryption (ABE) [30], [31] is a useful data encryption tool for enforcing access control policy via cryptographic technique. It makes sure that if a user has a legitimate credential, then he/she can decrypt a given ciphertext which is encrypted according to an access control policy [30], [31]. With the special features of ABE, Zheng et al. [19] proposed a new secure search service called attribute-based keyword search (ABKS). After that, [26], [27] were proposed to achieve extended functionalities (e.g., multi-keyword search) based on [19]. In ABKS, keywords are encrypted according to an access control policy, and a legitimate DU can generate trapdoors that can be used by server to search over the encrypted data. ABKS can maintain access control but also privacy-preserving keyword search in practice.

In order to enable DUs to make queries over multiple keywords at the same time, Golle et al. [28] proposed the concept of conjunctive keyword search. Later on, Park et al. [29] extended the notion into public key system. [26], [27] were put forth to achieve advanced functionalities in the multi-keyword search in public-key context. These schemes are denoted as attribute-based multiple keyword search (ABMKS) in this paper. However, the computation cost of keyword index generation is relatively high in [26], [27], which is growing linearly with the number of keyword embedded in a file. Recall that the complexity of keyword index ciphertext is $O(m)$, where $m$ is the number of keyword embedded in a file.

## 3 Preliminaries

We give a brief review of some basic primitives which will be used in this paper in Sections 3.1 and 3.2. We also define the main building blocks of our scheme in Sections 3.3-3.6.

### 3.1 Bilinear Map

Let $G$ and $G_T$ be two multiplicative cyclic groups of prime order $p$, $g$ be a generator of the group $G$, a bilinear mapping $e: G \times G \rightarrow G_T$ satisfies following properties [30]:

- Bilinearity: $e(g^a, g^b) = e(g, g)^{ab}$, $\forall g \in G, \forall a, b \in \mathbb{Z}_p^*$.
- Nondegeneracy: $e(g, g) \neq 1$.
- Computability: there is an efficient algorithm to compute $e(g^a, g^b)$, $\forall g \in G, \forall a, b \in \mathbb{Z}_p^*$

### 3.2 Access Tree

Let $T$ be an access tree [30] representing an access control policy. In $T$, each non-leaf node denotes a threshold gate and is described by its a threshold value and children notes. Let $\text{num}_v$ represent the number of children notes for a node $v$, and the children notes from left to right be labeled as $1, \ldots, \text{num}_v$. The $k_v$ ($k_v \leq \text{num}_v$) represents the threshold value for the node $v$, when $k_v=1$, the note $v$ is an OR gate; otherwise $k_v=\text{num}_v$, it is an AND gate. Each leaf node of $T$ is described by an attribute and a threshold value $k_v=1$. 

In order to easily understand the access tree, we give some few functions. Let parent \( (v) \) denote the parent of node \( v \). If \( v \) is a leaf node, att\( (v) \) represents the attribute associated with the leaf node \( v \). Let index\( (v) \) represent the label of the node \( v \), and \( T_v \) represent the subtree of \( T \) rooted at node \( v \).

\[ T_n(\gamma)=1 \]
denotes that the attribute set \( \gamma \) meets the access tree \( T_0 \). If \( v \) is a non-leaf node, we can compare \( T_n(\gamma) \) recursively as follows: compute \( T_n(\gamma) \) for all children \( v' \) of the node \( v \), if at least \( k_v \) children of the node \( v \) return 1, then \( T_n(\gamma)=1 \). If \( v \) is a leaf node, and att\( (v) \in \gamma \), then \( T_n(\gamma)=1 \).

### 3.3 Keyword Dictionary Tree

In this section, we first give the definition of the Keyword Dictionary Tree (KDT). Let \( W = \{w_1, ..., w_n\} \) be the keyword dictionary in the system, KDT denotes a full binary tree with \( n \) leaves, and the leaf node \( L_i \) be associated with a keyword \( w_i \) as in Fig. 1 (hereafter, we set \( n=8 \) in all the examples to easy understand). In a KDT tree, from left to right, from bottom to top, the nodes of the tree is labeled with 1, ..., \( 2n \) - 1. We give an example as in Fig. 1. The bottom left node of the KDT holds number 1, and the root node of KDT holds number 15.

### 3.4 Changed Dictionary Tree and the Path note

We call the KDT as an original dictionary tree (ODT), if the leaf \( L_i \) of it is labeled with the keyword \( w_i \). The changed dictionary tree (CDT) denotes that the positions of all the keywords in the ODT are changed. The CDT is defined as follows.

Let \( W' = \{w'_1, ..., w'_m\} \) be the keyword set for one file \( f_i \), and \( W=W/W' \) denote the difference of sets \( W \) and \( W' \), e.g., as shown in Fig. 2, \( W=\{w_1, ..., w_8\} \), \( W'=\{w_1, w_2, w_3, w_5, w_7, w_8\} \), then \( W=\{w_4, w_6\} \). DO will choose \( |W'| \) continuous leaf nodes of ODT, in which these leaf nodes are denoted as \( L \), and \( L^* \) is the remain of leaf nodes in ODT, e.g., as shown in Fig. 2, \( L=[3,4,5,6,7,8] \), and \( L^*=[1,2] \).

For each leaf node in \( L \) and \( L^* \), it will be randomly labeled with one keyword in \( W' \) and \( W \), respectively, as shown in Fig. 2, 1→w_6, 2→w_4, 3→w_7, 4→w_2, 5→w_1, 6→w_5, 7→w_8, and 8→w_3. Thus, all of the keyword labeled with the leaf node of ODT is changed. We state that there exists a permutation \( \sigma= [1, 6, 5, 2, 4][3, 7, 8] \), where \([1, 6, 5]\) indicates that leaf node 1 is labeled with keyword \( w_6 \), leaf node 6 is labeled with keyword \( w_5 \), and leaf node 5 is labeled with keyword \( w_1 \). The \([2, 4]\) and \([3, 7, 8]\) follow the same rule as \([1, 6, 5]\).

We note that CDT is generated by changing the keyword order of the ODT with the permutation \( \sigma \), and the \( \sigma \) has at least \( 2 \cdot m! \cdot (n - m)! = 2 \times (m \times (m - 1) \times ... \times 2 \times 1) \times [(n - m) \times (n - m - 1) \times ... \times 2 \times 1] \) different possibilities in total. Since \( n \) is a relatively large number, the \( \sigma \) will be very hard to be guessed (considering \( (n - m)! \) or \( m! \) is a very large number, e.g., \( n = 1000 \), \( m = 100 \)). As shown in Fig. 3, we present an example of generation of CDT by using \( \sigma= [1, 6, 5, 2, 4][3, 7, 8] \).

Let \( W^* = \{w^*_1, ..., w^*_m\} \) be a query keyword set, for each \( w^*_i \in W^* \), let path\( (w^*_i) = [\Omega_{w_i,1}, ..., \Omega_{w_i,t}] \) (where \( l = 1+\log n \) be the path nodes of the keyword \( w^*_i \), which starts from the leaf node labeled with \( w_i \) to the root node in CDT, e.g., in Fig. 2, path\( (w_4) = [2, 9, 13, 15] \).

### 3.5 Subset Cover

In this section, we give the definition of the subset cover for a keyword set \( W' \). For the keyword set \( W' \) in the CDT, with the subset cover technique [32], DO selects root nodes of the minimum cover sets in the CDT tree that can cover all of the leaf nodes in \( W' \), denoted as \( \text{cover}(W') = [\Omega_1, ..., \Omega_t] \), where \( t \) is the number of node in cover\( (W') \) and \( t \leq 2 \cdot \log (n/2) \), e.g., as in Fig. 2. When \( W' = \{w_1, w_2, w_3, w_5, w_7, w_8\} \), we have the cover\( (W') = [10, 14] \). For each keywords \( W' \), we will prove that \( t \leq 2 \cdot \log (n/2) \) as follows.
As shown in Fig. 5, in case 2, the keyword \( W' \) are continuously labeled with both right and left half of CDT, the keywords in \( W' \) that are labeled with the left half leaf nodes of CDT denoted as \( W_{left} \), and the keywords in \( W' \) that are labeled with the right half leaf nodes of CDT denoted as \( W_{right} \).

Consider the keyword \( W_{left} \) and \( cover(W_{left}) \) (where \( t_{left} \) denotes the number of nodes in \( cover(W_{left}) \)). Because \( W' \) are continuously labeled with the leaf nodes in CDT, we note that \( t_{left} \leq 1 + \log(2^k/2) = k \) (As shown in Fig. 5, the number of cover note in the black rectangle is log \((2^k/2)\), and the number of cover note in the red rectangle is 1). Similarly, we have \( t_{right} \leq 1 + \log(2^k/2) = k \). Therefore, we have \( t = t_{left} + t_{right} \leq k + k = 2k = 2 \log(2^{k+1}/2) = 2 \log(n/2) \).

Therefore the Theorem 1 is proved. \( \square \)

**Theorem 2.** In the CDT, for each keyword set \( W' \), the number of node in the subset \( cover(W') = t \leq m \), where \( t \) is the number of node in \( cover(W') \), \( m \) is the number of keyword in keyword set \( W' \).

Proof. As the example shown in Fig. 2, recall that the keywords in \( W' \) are continuously labeled with the leaf nodes in CDT, then there must exist two keywords that have the same cover node in CDT. Thus, it is easy to prove that \( t \leq m \). \( \square \)

**Theorem 3.** From Theorem 1 and Theorem 2, we have that \( t \leq \min\{2 \log(n/2), m\} \)

### 3.6 Key Idea of Keyword Index Ciphertext Generation and Match

In this section, we present the idea of keyword index ciphertext generation, and the process of whether a query keyword set \( W^* \) matches the keyword set \( W' = [w'_1, \ldots, w'_m] \) embedded in a file.

When a DO wants to share a file, and the keywords set \( W' \) is the keyword set embedded in it. First, DO generates a permutation \( \sigma \) with \( W' \) as in the Section 3.4, and obtains the cover(\( W' \)) as in the Section 3.5. Then DO chooses an access policy tree \( T \), and encrypts the permutation \( \sigma \) based on the access policy tree by using ABE scheme.

\[
\text{cover}(W') = [\Omega_1, \ldots, \Omega_t]
\]

Note that DO will use all the elements of cover(\( W' \)) to build the keyword index ciphertext instead of \( W' \). We will give more details of it in the Encrypt algorithm in Section 5. Thus the complexity of computation and size is reduced from \( m \) to \( t \). As in the Theorem 3, we have proved that \( t \leq \min\{2 \log(n/2), m\} \).

Let \( W^* = [w^*_1, \ldots, w^*_m] \) be the query keyword set chosen by a data user. If the data user’s attributes set satisfies the access policy tree \( T \), then he/she can get the permutation \( \sigma \), which means that he/she is allowed to generate the CDT as in the Section 3.4. Thus, for each \( w^*_i \in W^* \), the data user can get the path(\( w^*_i \)) from the CDT.

\[
\text{path}(w^*_i) = [\Omega_{w^*_i,l}, \ldots, \Omega_{w^*_i,1}], \quad l = 1 + \log n
\]

To verify whether the query keyword set \( W^* \subseteq W' \), we need to check that whether there is an element in both \( \text{path}(w^*_i) \) and \( \text{cover}(W') \), for each \( w^*_i \in W^* \). We present this theorem by the following formula.
\( W^* \subseteq W' \iff \text{path}(w^*_i) \cap \text{cover}(W') \neq \emptyset, \forall w^*_i \in W^*. \)

Otherwise, if \( W^* \not\subseteq W' \), then there exists one keyword \( w_1 \) satisfying \( \text{path}(w_1) \cap \text{cover}(W') = \emptyset \). For example, as in Fig. 2, if \( W' = [w_1, w_2, w_3, w_5, w_7, w_8], W^* = [w_1, w_2, w_5, w_8] \), we see that

\[
\begin{align*}
\text{path}(w_1) &= [5, 11, 14, 15], \\
\text{path}(w_2) &= [4, 10, 13, 15], \\
\text{path}(w_5) &= [6, 11, 14, 15], \\
\text{path}(w_8) &= [7, 12, 14, 15], \\
\text{cover}(W') &= [10, 14].
\end{align*}
\]

Note that

\[
\begin{align*}
\text{path}(w_1) \cap \text{cover}(W') &= 14, \\
\text{path}(w_2) \cap \text{cover}(W') &= 10, \\
\text{path}(w_5) \cap \text{cover}(W') &= 14, \\
\text{path}(w_8) \cap \text{cover}(W') &= 14.
\end{align*}
\]

Remark. Using the above method to generate keyword index ciphertext, the complexity of keyword index ciphertext and the computation complexity is at most \( O(2\log(n/2)) \) for the worst case, but we state that the complex is \( O(1) \) for the best case (In this case, the number of node in the cover\( (W') \) is only 1), where \( n \) is number of the keywords in keyword dictionary \( W \). It is to say the number of the keyword index ciphertext can be reduced to only one.

4 Problem Formulations

In this section, we present the system and threat models, the scheme construction, and the security model.

4.1 System and Threat Models

In this section, we present the security model of our scheme. which consists of eight algorithms.

- **Setup(\( \lambda \)).** The Setup algorithm takes as input a security parameter \( \lambda \). It initializes the global system parameter, and outputs the master key \( MSK \) and the public key \( PK \).

- **KeyGen(\( PK, MSK, S \)).** The KeyGen algorithm takes as input data user’s attribute set \( S \), the authority uses the \( MSK \) to generate the private key \( SK \) for the data user.

- **Encrypt(\( PK, T, ODT, F, W' \)).** The Encrypt algorithm takes as input files set \( F \) and the keyword dictionary \( W \). For each file \( f \in F \), DO generates the permutation \( \sigma \) by using the keyword set \( W' \) corresponding to the file \( f \), then encrypts the file \( f \) by using the symmetric key \( K \) to get the ciphertext \( C \), encrypts the \( \sigma \) and \( K \) by using ABE to get the ciphertext \( C' \), generates the encrypted index \( I' \) according to \( W' \). At last sends the ciphertext \( CT = \{C, I_1, I_2\} \) to the CSP.

- **GenTK(\( PK, SK \)).** The GenTK algorithm takes as input public key \( PK \) and the private key \( SK \) for user’s attribute set \( S \), user generates the transformation key \( TK \) and the corresponding retrieving key \( RK \).

- **Transform(\( CT, TK \)).** The Transform algorithm takes as input the ciphertext \( CT \) and the transformation key \( TK \), it outputs a partially decrypted ciphertext \( CT' = \{CT'_1, CT'_2\} \).

- **Decrypt(\( CT'_1, RK \)).** The Decrypt algorithm takes as input the transformed ciphertext \( CT'_1 \) and the retrieving key \( RK \), it outputs the permutation \( \sigma \) and the symmetric key \( K \).

- **Trapdoor(\( PK, SK, RK, W^*, ODT, \sigma \)).** The trapdoor algorithm takes as input the public key \( PK \), private key \( SK \) and corresponding key \( RK \), queried keyword set \( W^* \), original dictionary tree \( ODT \), the permutation \( \sigma \), it outputs the trapdoor \( T_{W^*} \) according to the query keyword \( W^* \) and the data user’s attribute set \( S \).

- **Retrieve(\( PK, CT'_1, T_{W^*} \)).** The Retrieve algorithm takes as input the ciphertext \( CT' \) and the trapdoor \( T_{W^*} \). CSP checks whether \( T_{W^*} \) satisfies the ciphertext \( I_1 \) and the encrypted index \( I_2 \). If it holds, then returns the corresponding \( C \) to user, otherwise, outputs \( \perp \).

4.3 Security Model

In this section, we present the security model of our scheme. As described in the threat model, only the CSP is honest-but-curious. Intuitively, indistinguishability against chosen-keyword attack (IND-CKA) means that the CSP (an adversary \( A \)) can learn nothing information about keyword
set plaintext of the keywords set ciphertext except for the search tokens and the result. We present the security model by utilizing the indistinguishability against chosen-keyword attack (CKA) game as follows.

### Definition 1. IND-CKA Game:

- **Setup**: The challenger C runs the setup algorithm to generate the public parameters PK and the master key MSK, and gives the public parameters PK to the adversary A. Note that the master key MSK is kept by the challenger C. The adversary A chooses an access tree T, which is sent to the challenger.

- **Phase 1**: A can adaptively query the following oracles for polynomial time, and the challenger C initializes an empty keyword list \( L_{kw} \) and an empty set D.

1. \( O_{KeyGen}(S) \): On input a set of attributes S, the challenger C runs the KeyGen algorithm to get \( SK_S \) and sets \( D = D \cup S \). It then returns it to adversary A.

2. \( O_{GenTK}(SK) \): On input a set of attributes S, if \( S \in D \), the challenger C runs the GenTK algorithm to get \( TK_S \). Otherwise, the challenge runs the KeyGen algorithm to get \( SK_S \), and runs GenTK to get \( TK_S \). It then returns the \( TK_S \) to adversary A.

3. \( O_{Trapdoor}(SK, W^*) \): On input a set of keyword \( W^* \) and the SK, the challenger C runs the Trapdoor algorithm to get \( T_{W^*} \) and sets \( L_{kw} = L_{kw} \cup W^* \), if the attributes set S satisfies the policy tree \( T \). It then returns it to adversary A.

- **Challenge**: A randomly chooses two keyword set \( W_1^* \) and \( W_2^* \), where \( W_1^*, W_2^* \notin L_{kw} \), it means that \( W_1^* \), \( W_2^* \) cannot be queried in \( O_{Trapdoor} \). Then, the challenge T picks a random \( b \in \{0,1\} \) and encrypts \( W_b^* \) as \( CT^b \) by using Encrypt algorithm. Finally, C returns \( CT^b \) to the adversary A.

- **Phase 2**: A continues to query the oracles as in Phase 1, but the restriction is that \( (S, W_1^*) \) and \( (S, W_2^*) \) cannot be the input to \( O_{Trapdoor} \) if the attribute set S satisfies the access policy \( T \).

### Definition 2. Our scheme is IND-CKA secure if the advantage of any PPT A winning the above IND-CKA game is negligible.

Besides, our scheme is selectively IND-CKA secure if an Int stage is added before the Setup algorithm where the adversary A claims the two keyword sets \( W_1^*, W_2^* \) which is sent to be attacked.

### 5 Our Design

We present some notations used in our construction in the TABLE 1 and our design below.

- **Setup(\( \lambda \))→(PK, MSK)**: This algorithm is executed by authority, given a security parameter \( \lambda \), the authority first chooses a bilinear group \( G \) of prime order \( p \) with generator \( g \). It then chooses two random numbers \( \alpha_1, \alpha_2 \in \mathbb{Z}^*_p \) and two hash functions \( H_1: \{0,1\}^* \rightarrow \mathbb{G} \), \( H_2: \{0,1\}^* \rightarrow \mathbb{Z}^*_p \). Finally, it generates the public key PK and master key MSK as follows:

\[
PK = \{g, g^h, g^{\sigma}, e(g, g)^{\alpha_1}\}
\]

\[
MSK = \{\alpha_2, g^{\alpha_1}\}
\]

- **KeyGen(\( PK, S, MSK \))→SK**: This algorithm is executed by authority, given a DU’s attribute set \( S \), the authority first chooses a random number \( r \in \mathbb{Z}^*_p \) and randomly chooses \( w_i \in \mathbb{Z}^*_p \) for each attribute \( w_i \in S \). Finally, it outputs DU’s private key SK as follows:

\[
SK = \{D = g^{\langle 1 \rangle} \alpha_1 + r / \alpha_2, \forall i \in S: D_i = g^{H_2(w_i)}H_1(w_i)^r, D_i' = g^{r_i}\}
\]

- **Encrypt(\( PK, T, ODT, F, W \))→CT**: This algorithm is executed by DO. Let \( F = \{f_1, ..., f_N\} \) be the file set to be shared, in order to easy understand, we encrypt one file \( f \in F \) to explain the Encrypt algorithm, let the \( W = [w_1, ..., w_m] \) be the keyword dictionary, and the \( W' = [w_1', ..., w_m'] \) be the keyword set for \( f \).

DO will encrypt the \( f \) under the corresponding symmetric key \( K \) to generate the ciphertext \( C \) (e.g., Using AES to encrypt the \( f \), this is out of the scope of our discuss).

After that, DO will generate the \( \sigma \) and \( \text{cover}(W') \) for file \( f \) as follows.

1. As in Section 3.4, DO will randomly generate a permutation \( \sigma \) by using \( W' \).

2. As in Section 3.4, DO will generate the CDT by using \( \sigma \) and ODT.

\[
\text{CDT} \rightarrow_{\sigma} \text{ODT}
\]

3. As in Section 3.5, DO will get the \( \text{cover}(W') \) corresponding to \( W' \).

\[
\text{cover}(W') = [\Omega_1, ..., \Omega_t]
\]

As shown in the Theorem 3, we state that \( t \leq \text{Min}(2 \log (n/2), m) \). For example, as shown in the Fig. 2, when \( W' = [w_1, w_2, w_3, w_5, w_7, w_8] \), and \( \sigma = \)
Then, DO computes the ciphertext $I_1$ and $I_2$ for the file $f$ as follows:

1) For each node $v$ in the in the access policy tree $T$, choose a polynomial $q_v$ in a top-down manner, the degree $d_v$ of $q_v$ is $k_v - 1$, where $k_v$ is the threshold value of the node $v$. For the root node $R$ of $T$, randomly choose $s \in \mathbb{Z}_p^*$ and set $q_R(0) = s$, then randomly choose $d_R$ other points to build the polynomial $q_R$. For the non-root node $v$, set $q_v(0) = q_{parent(v)}(index(v))$ and randomly choose $d_v$ other points to build the polynomial $q_v$.

Let $Y$ be the set of leaf nodes in the access tree $T$ and compute $I_1 = \{I_\sigma||K, \{\theta, \theta_y, \theta_y^x\}\}$

$$I_\sigma||K = (\sigma||K) \cdot e(g,g)^{\alpha_1s}.\theta = h^{s}, \theta_y = g^{q_y(0)}, \quad \theta_y^x = H_1(att(y))_{\theta_y(0)}, \forall y \in Y.$$ 

2) Compute $I_2 = \{\text{cover}(W'), \{I_{\Omega_i}\}_{i=1}^t\}$ with the permutation $\sigma$ and the cover $(W') = [\Omega_1, ..., \Omega_t]$.

Then output $CT = \{C, I_1, I_2\}$ for the file $f$.

- **GenTK($PK, SK$)$\rightarrow TK$:** This algorithm is executed by DU, it takes into the public key $PK$ and the user’s private key $SK = \{D = g^{(\sigma_i+\tau)/2}, D_i = g^{H_1(i)^{\tau}}, D_i' = g^{\gamma}\}$ for a set of attributes $S$. It chooses a random value $u \in \mathbb{Z}_p^*$ and computes transformation key $TK$ and the corresponding retrieving key $RK$:

$$TK = \{S, D^* = D^u, D_i^* = D_i'^u, D_i'^* = D_i'^{u^2}\}$$

$$RK = u$$

- **Transform($TK, CT$)$\rightarrow (CT'\&\perp)$**: This algorithm is executed by cloud service provider (CSP), after getting user’s attributes set $S$ from $TK$, then CSP checks whether the $S$ can meet $T$. If $S$ can not meet $T$, it outputs $\perp$; otherwise, it runs the algorithm as follows:

1) If the node $x$ is a leaf node in the $T$, we let $i = \text{att}(x)$ and define as follows: if $i \in S$, then compute $\theta_x$ as follows:

$$\theta_x = \frac{e(D_i^x, \theta_x)}{e(D_i^*, \theta_x)} = \frac{e(g^{\gamma H_1(i)^{\tau}}, g^{q_{x}(0)})}{e(g^{\gamma H_1(i)^{\tau}}, g^{q_{x}(0)})} = e(g,g)^{\gamma u q_x(0)} (1)$$

2) If the node $x$ is a no-leaf node in the $T$, we get the $\theta_x$ by computing $\theta_x'$ in a recursive manner, where $x'$ is the children nodes of $x$.

The $S_x$ is an arbitrary $k_x$ set of children nodes $x$, if there exists no such a set, set $\theta_x = \perp$; otherwise, compute $\theta_x$ as:

$$\theta_x = \prod_{x' \in S_x} \theta_x^{D_i^x, \theta_x} = \prod_{x' \in S_x} (e(g,g)^{\gamma u q_x(0)})^{\Delta_i, S_x(0)} = \prod_{x' \in S_x} (e(g,g)^{\gamma u q_{parent}(x')}:\text{index}(x'))^{\Delta_i, S_x'(0)} = \prod_{x' \in S_x} (e(g,g)^{\gamma u q_{x}(i)})^{\Delta_i, S_x'(0)} = e(g,g)^{\gamma u q_x(0)}$$

where $i = \text{index}(x'), S_x' = \{\text{index}(x') : x' \in S_x\}$.

If the tree is satisfied by $S$, we can get that $A = \theta_{root} = e(g,g)^{\gamma u q(0)} = e(g,g)^{\gamma u s}$, and compute the partially-decrypted ciphertext $pdc$ as

$$pdc = e(\theta, D^*) / A$$

$$= e(h^s, g^{(\alpha_1 + \tau)u/2}) / e(g,g)^{\gamma u s} = e(g,g)^{\gamma su} (3)$$

Then output $CT' = \{CT_1', CT_2\}$, and return $CT_1'$ to DU.

- **Decrypt($RK, CT_1'$)$\rightarrow (\sigma||K)$**: This algorithm is executed by DU, on receiving the $CW'$ from CSP, then obtain the $\sigma$ and $K$ by using retrieve key $RK$ as

$$I_{\sigma||K}/pdc \cdot \pi_{\overline{K}} = (\sigma||K) \cdot e(g,g)^{\alpha_1s}/e(g,g)^{\alpha_1su} = \sigma||K$$

Thus, the DU gets the permutation $\sigma$ and the symmetric key $K$ corresponding to the file $F$.

- **Trapdoor($PK, SK, RK, W^*, ODT, \sigma$)$\rightarrow TK_W^*$**: When a user wants to issue a search query according to keyword set $W^* = [w_1^*, ..., w_m^*]$ , where $m$ is the number of the keywords included in $W^*$. After getting the permutation $\sigma$ and $K$, DU will generate the CDT and $\{\text{path}(w_i^*)\}_{i=1}^m$ with ODT and $\sigma$ as follows:

1) $\text{CDT}_e^m\&\text{ODT}$ as in Section 3.4.

2) For each $w_i^* \in W^*$, obtain the path nodes $\text{path}(w_i^*)$ as in Section 3.4.

$$\text{path}(w_i^*) = [\Omega_{w_i^*,1}, ..., \Omega_{w_i^*,t}], t = 1 + \log n.$$ 

Then compute $T_1$ and $T_2$ as follows:

$$T_1 = \{S, D^* = D_{RK}, D_i^* = D_{i,RK}, D_i'^* = D_i'^{RK}\} = \{S, D^* = D^u, D_i^* = D_i'^u, D_i'^* = D_i'^{u^2}\}$$

$$T_2 = \{\text{path}(w_i^*), T_{\Omega_{w_i^*,j}} = H_2(\Omega_{w_i^*,j}), \text{path}(w_i^*)\} = \{\text{path}(w_i^*), T_{\Omega_{w_i^*,j}} = H_2(\Omega_{w_i^*,j}), \text{path}(w_i^*)\} \quad u$$

for $i \in [1, m^*], j \in [1, 1+\log n]$. 

\[1, 6, 5\] [2, 4] [3, 7, 8], then the cover($W^*$) = [10, 14].
Note that $T_1 = TK$ is computed by DU as in the GenTK phase.
Finally, return $T_{W^*} = \{T_1, T_2\}$. 

- **Retrieve**($PK$, $CT^*, T_{W^*}$) → $(C \& \bot)$: This algorithm is executed by CSP, and it works as follow. If $S$ does satisfy $T$, the CSP will terminate the search process; otherwise, the CSP works as follows.

For each path($w^*_i$) for the keyword $w^*_i \in W^*$, if there exists one case that $\text{cover}(W^*) \cap \text{path}(w^*_i) = 0$, CSP breaks the retrieve and outputs $\bot$. It is because that $W^* \subseteq W^* \iff \text{path}(w^*_i) \cap \text{cover}(W^*) \not= 0$, $\forall w_i \in W^*$ as described in Section 3.6.

Otherwise, we let $\Omega_i = \text{cover}(W^*) \cap \text{path}(w^*_i)$, $i \in [1, m^*]$. Check

$$i = m^* \prod_{i=1}^{i=m^*} \Omega_i \prod_{i=1}^{i=m^*} pdc_i \cdot \bar{\Omega}_i = \prod_{i=1}^{i=m^*} e(g, g^{\alpha_1 H_2(\Omega_i || \sigma || K)})$$ (4)

If it does not hold, return $\bot$; otherwise, it means that Trapdoor $T_{W^*}$ matches the keyword index ciphertext $I$, then send the corresponding C to the user. Once gaining all the search results from CSP, the user can decrypt them by using the corresponding symmetric key $K$.

**Correctness analysis.** If the user’s attribute set $S$ satisfies the access policy tree $T$ and the queried keyword set satisfies $W^* \subseteq W$, we have that

$$i = m^* \prod_{i=1}^{i=m^*} \Omega_i \prod_{i=1}^{i=m^*} pdc_i \cdot \bar{\Omega}_i = \prod_{i=1}^{i=m^*} e(g, g^{\alpha_1 H_2(\Omega_i || \sigma || K)})$$ (5)

If $\text{path}(w^*_i)$ represents the path nodes set, which consists of the nodes originate from the leaf node associated with the keyword $w^*_i$ up to the root node of the tree CDT, then computes $\bar{T}_{W^*}$, $i \in [1, m^*], j \in [1, \log n]$, and sets

$$T_2 = \{\Omega_0, T_{W^*}\}. \text{If the attributes set } S \text{ satisfies the policy tree } T^*, \text{the challenger adds the keyword set } W^* \text{ to the keyword list } L_{kw}.$$

**Challenge phase** : The $A$ gives two keyword sets $W_0$, $W_1$ to be challenged on, where $W_0$, $W_1 \not= L_{kw}$ and the length of $W_0$ is equal to $W_1$, the challenger randomly chooses $s \in Z_p$, a symmetric key $K$ and computes secret shares of $s$ for each leaves in $T^*$. The challenger chooses $\lambda \leftarrow \{0, 1\}$ and generates a permutation $\sigma_\lambda$ corresponding to $W^*_\lambda$. For $\lambda = 0$, it runs the Encrypt algorithm to generate the ciphertext corresponding to keyword set $W^*_\lambda$ and outputs

$$I_{\sigma_\lambda} = (\sigma_\lambda || K) e(g, g)^{\alpha_1 H_2(\Omega_i || \sigma || K)} \theta = h^n;$$

$$\{I_i = e(g, g)^{\alpha_1 H_2(\Omega_i || \sigma || K)} \}_{i \in [1, l]},$$

$$\{\theta_y = g^{\eta_y} \cdot \theta_y = H_1(\text{att}(y))^{\eta_y} \}_{Y \in Y} \text{ by selecting } \eta \in Z_p.$$ otherwise, it outputs

$$I_{\sigma_\lambda} = (\sigma_\lambda || K) e(g, g)^{\alpha_1 \theta \lambda H_2(\Omega_i || \sigma || K)} \theta = h^n;$$

$$\{I_i = e(g, g)^{\alpha_1 H_2(\Omega_i || \sigma || K)} \}_{i \in [1, l]},$$

$$\{\theta_y = g^{\eta_y} \cdot \theta_y = H_1(\text{att}(y))^{\eta_y} \}_{Y \in Y} \text{ by selecting } \eta \in Z_p.$$ 

**Phase 2** : This phase is like Phase 1, but the restriction is that $W^*_0$, $W^*_1$ have not been issued in Phase 1.

**Guess** : The adversary $A$ outputs a guess for $\lambda \in \{0, 1\}$. If $\lambda = \lambda$, $A$ wins the IND-CCA game; otherwise, it fails. We can note that if $A$ can construct $e(g, g)^{\alpha_1 H_2(\Omega_i || \sigma || K)}$ by using the term $e(g, g)^{\eta}$ contained in the aforementioned oracles, then $A$ can use it to distinguish $e(g, g)^{\eta}$ from...
The bilinear pairing operation $e$, exponentiation operation in group $G$, the number of keyword in $G$, the number of a DU’s attributes, the element length in $G_T$, the number of nodes in the cover $T$, map a bit-string to an element of $G_T$, to the access tree. Therefore, we prove that $\mathcal{A}$ cannot gain non-negligible advantage in the IND-CKA game. As $\alpha_1 s$ can be rebuilt by using $(\alpha_1 + r^s) / \alpha_2$, $q_0(0)$, $\alpha_2 s$ due to $((\alpha_1 + r^s) / \alpha_2) s = \alpha_1 s + r^s s$, $\mathcal{A}$ needs to cancel $r^s s$, which needs to use the terms $r^s_1$, $r^s_2$, $q_0(0)$, $r^s_3 q_0(0)$. Because of $q_0(0)$ is the secret share of $s$ according to the access tree $T^*$. But the adversary $\mathcal{A}$ cannot rebuild $r^s$ for that the terms outlined above can only be rebuilt only if the attribute set $S$ can meet $T^*$. Therefore, we proved that $\mathcal{A}$ gains a negligible advantage in the IND-CKA game. It is to say that our scheme is secure against IND-CKA. This completes the proof.

### 7 Performance Evaluation

In this section, we present the efficiency analysis for our scheme in terms of both complexity and actual execution time, and further compare our scheme with the state-of-the-art PAB-MKS [27]. TABLE 2 defines the notation used in comparison.

In the theoretical analysis, we mainly show the computational and storage cost complexity in TABLE 3 and TABLE 4, respectively. We mainly consider the time-consuming operations, namely, bilinear pairing operation $P$, exponentiation operation $E_G$ in group $G$, exponentiation operation $E_{G_T}$ in group $G_T$, hash operation $H_1$ and $H_2$. We do not consider the multiplication operations because they are much more lightweight than the above operations.

#### 7.1 Theoretical Comparison

In TABLE 3, we present the comparison of computation cost under the same access control policy tree $T$. We observe that the complexity of KeyGen in our scheme is the same as that of the PAB-MKS [27].

In the PAB-MKS scheme, the encryption cost is more expensive than that of our scheme. Specifically, the former scheme computation cost is $(m + 2|Y| + 3) \cdot E_G + |Y| \cdot H_1 + m \cdot H_2$, while in our scheme the computation cost is $(t + 1) \cdot E_{G_T} + (2|Y| + 1) \cdot E_G + |Y| \cdot H_1 + t \cdot H_2$. We note that $t \leq 2 \cdot \log(n/2)$, where $n$ is the number of keyword in the keyword dictionary $W, m$ is the number of keyword in $W'$ embedded in one file, then $t \ll m$, e.g., when $n = 1000$, $t \leq 2 \cdot \log 1000/2 \leq 18$, and the minimum keyword number in a file is about $m = 100$.

We state that the computation cost of Trapdoor and Retrieve in our scheme is also less than that of PAB-MKS. The computation cost of Trapdoor generation in PAB-MKS and our scheme are $(2s + 3) \cdot E_{G_T} + m^* \cdot H_2$ and $(2s + 1) \cdot E_G + (m^* \cdot \log n) \cdot H_2$, respectively.

The computation cost of retrieve in PAB-MKS and our scheme are $(2s + 4) \cdot P + s \cdot E_{G_T}$ and $(2s + 1) \cdot P + (s + 2m^* - 1) \cdot E_{G_T}$, respectively.

In TABLE 4, we present the comparison of storage cost. With the same reason shown as in TABLE 3, the storage cost of keyword index ciphertext (the output of Encrypt algorithm) in our scheme is less than that in PAB-MKS scheme, while the storage cost of KeyGen in our scheme is same as that of PAB-MKS. Our scheme has higher storage in Trapdoor, but it is acceptable and it is only one-time operation.

#### 7.2 Experimental Performance

To present the practicability of our scheme in practice, we implement the scheme and PAB-MKS [27] with real-world dataset using Python language, and further run the experiment tests for 100 times, in which the dataset includes 1000 distinct keywords extracted from 500 PDF files (academic papers) provided by the Google Scholar. The maximum number of the keywords in a file is 200 while the minimum is 100. Our experimental platform is on Ubuntu 16.04 LTS with Intel Core i3 Processor 4170 CPU @3.70GHZ with 10.0 GB of RAM. Since these two schemes are highly dependent on the basic cryptographic operations in the pairing computation, we implement PBA-MKS and our scheme in software based on the libfenc library [33], using a 224-bit ($|G| = |G_T| = 224$ bits, $Z_p = 224$ bits) MNT elliptic curve from the Stanford Pairing-Based Crypto library.
To bring convenience in comparison, in PAB-MKS and our scheme, we encrypt the keyword sets of each file by using the same access control policy \( T \). For example, the policy tree \( T \) is “AND” access tree: \( (A_1 \text{ AND } A_2 \text{ AND}... \text{AND} A_l \text{|Y|}) \), where \( A_i \) is an attribute. We set the number of DU’s attribute \( s \in [10, 20, 30, 40, 50] \), the number of leaf node in the access policy tree \( |Y| \in [20, 40, 60, 80, 100] \), the number of keyword in each files \( m=100 \), and the number of keyword in the query \( m^* \in [2, 4, 6, 8, 10] \).

Fig. 7a presents the computation time cost of ciphertext generation which is executed by DO. As described in TABLE 3, the ciphertext generation time cost of the scheme PAB-MKS is affected by two factors, the number of keyword embedded in a file \( m \) and the number of leaf nodes \( |Y| \) in policy tree \( T \), while the ciphertext generation time is affected by \( t \) and \( |Y| \) in our scheme. We further set \( m = 100 \) and \( n = 1000 \) in the Encrypt phase. Because \( t \leq 2\log(n/2) \leq 18 \), and then \( t < m \), thus, we have that the cost time of ciphertext generation in our scheme is efficient than PAB-MKS. When \( |Y| = 60 \), the computation cost time is 131ms for us, while the PAB-MKS scheme needs 946ms to generate a ciphertext.

Fig. 7b presents the computation time cost of generating a trapdoor which is executed by DU. As described in TABLE 3, the trapdoor generation time cost of the scheme PAB-MKS and our scheme is affected by two factors: \( m^* \) and \( s \). The PAB-MKS scheme and our scheme need \((2s+3) \cdot E_G + m^* \cdot H_2, (2s+1) \cdot E_G + (m^* \cdot \log n \cdot H_2)\) in trapdoor generation phase, respectively. Because the hash operation \( H_1 \) is much more efficient than the exponentiation operations [19], then the hash operation almost can be ignored. Thus, the trapdoor generation cost in our scheme is efficient than that of PAB-MKS. As Fig. 7(b), we can see that when \( m^* = 2 \) and \( m^* = 10 \), the our cost of generating trapdoor is efficient than that in PAB-MKS scheme, respectively. For example, when \( s = 50 \), \( m^* = 10 \), our scheme (requiring 809 ms) outperforms the PAB-MKS scheme (1186 ms) by around 300 ms.

Fig. 7c presents the computation time cost of retrieving the ciphertext. As described in TABLE 3, the time cost of retrieve algorithm in PAB-MKS and our scheme are \((2s+4) \cdot P + s \cdot E_{Gr}, (2s+1) \cdot P + (s+1) \cdot E_{Gr}\), respectively. Since one time exponentiation computation is efficient than one time pairing operation under the same security condition,
we can state that our scheme is efficient than PAB-MKS in retrieving process. For example, when \( s = 50 \), \( m^* = 6 \), our scheme only takes 74.636748 ms.

As described in TABLE 4, the storage cost of KeyGen algorithm in PAB-MKS and our scheme are \((2s + 1) \cdot |G|, (2s + 1) \cdot |G|\), respectively. As shown in Fig. 7d, the storage cost of KeyGen algorithm in our scheme is the same as that in PAB-MKS.

Fig. 7e and Fig. 7g present the ciphertext size of PAB-MKS and our scheme. As the analysis in TABLE 4, the number of keyword index ciphertext of PAB-MKS is affected by \( m \) and \( |Y| \), while ours is affected by \( t \) and \( |Y| \). Due to \( t \leq 2 \cdot \log (n/2) \), as shown in Fig. 7e, when setting \( m = 100 \), as the number of keyword index ciphertext in our scheme decreases, our scheme outperforms the PAB-MKS in terms of storage cost in Encrypt algorithm. For instance, when setting \( n = 1000, m = 100, |Y| = 60 \), the ciphertext length of PAB-MKS is 12.19 KB, while ours is 8.42 KB. As shown in Fig. 7g, when setting \(|Y| = 20\), with the increase of keyword number \( m \), the number of keyword index ciphertext in our scheme is constant, while that of PAB-MKS is linearly growing with \( m \).

Fig. 7f presents the trapdoor size of the two schemes. As the analysis in TABLE 4, compared with PAB-MKS, the trapdoor size in our scheme is slightly more than PAB-MKS due to the extra item \((m^* \cdot \log n) |Z_p|\). For example, when setting \( s = 50, m^* = 10 \), the trapdoor size of PAB-MKS and ours are 12.875 KB and 14.578 KB, respectively. Then our scheme is still acceptable in practice since the Trapdoor algorithm is a one-time cost.

8 CONCLUSION

In this paper, we’ve proposed an efficient ABMKS with short-size ciphertext of keyword index which provides secure multi-keyword search service with fine-grained access control. Its number of search index is very small, being independent on the number of underlying keyword in a file. The formal security analysis shows that our scheme is secure. Moreover, the performance evaluation demonstrates that the scheme is efficient in terms of both the computation and communication overhead in practice.

ACKNOWLEDGMENTS

This work is supported by NSFC (Grant Nos. 61672110, 61671082, 61976024, 61972048)

REFERENCES


[33] Green, M., Akinyele, A., & Rushanan, M. (2004). libfenc: The functional encryption library. Available from http://code.google.com/p/libfenc. Baodong Qin received the B. Sc. degree. Zhidan Li received his bachelor’s degree at Zhengzhou University in 2013. Now he is doing research in the Institute of Network and Technology at BUPT and his interests are focused on the network security and cryptography protocols. E-mail: zhidanli@bupt.edu.cn

Wei Yin Wei Yin, received the B.S. degree in Mathematics and Applied Mathematics from Huaibei Normal University, Huaibei, Anhui, China, in 2012, and the Ph.D degree in cryptography from Beijing University of Posts and Telecommunications, Beijing, China, in 2019. His research interests include public key cryptography, lattice cryptography, and provable security.

Hua Zhang Hua Zhang received the B.S. degree in telecommunications engineering from the Xidian University in 1998, the MS degree in cryptography from Xidian University in 2005, and the Ph.D degree in cryptography from Beijing University of Posts and Telecommunications in 2008. Now she is a lecturer of Beijing University of Posts and Telecommunications. Her research interests include cryptography, information security and network security. E-mail: zhanghua_288@bupt.edu.cn

Wei Yin, received the B.S. degree in Mathematics and Applied Mathematics from Huaibei Normal University, Huaibei, Anhui, China, in 2012, and the Ph.D degree in cryptography from Beijing University of Posts and Telecommunications, Beijing, China, in 2019. His research interests include public key cryptography, lattice cryptography, and provable security.

Qiaoyan Wen Qiaoyan Wen received the B.S. and M.S. degrees in Mathematics from Shaanxi Normal University, Xi’an, China, in 1981 and 1984, respectively, and the Ph.D degree in cryptography from Xidian University, Xi’an, China, in 1997. She is a professor of Beijing University of Posts and Telecommunications. Her present research interests include coding theory, cryptography, information security, internet security and applied mathematics. E-mail: wqy@bupt.edu.cn

Kaitai Liang Kaitai Liang received the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, in 2014. He is currently an Assistant Professor with the Department of Computer Science, University of Surrey, U.K. His research interests are applied cryptography and information security, in particular, encryption, network security, big data security, privacy enhancing technology, blockchain, lattice-based crypto and security in cloud computing. E-mail: k.liang@surrey.ac.uk

Wei Yin, received the B.S. degree in Mathematics and Applied Mathematics from Huaibei Normal University, Huaibei, Anhui, China, in 2012, and the Ph.D degree in cryptography from Beijing University of Posts and Telecommunications, Beijing, China, in 2019. His research interests include public key cryptography, lattice cryptography, and provable security.

Hua Zhang Hua Zhang received the B.S. degree in telecommunications engineering from the Xidian University in 1998, the MS degree in cryptography from Xidian University in 2005, and the Ph.D degree in cryptography from Beijing University of Posts and Telecommunications in 2008. Now she is a lecturer of Beijing University of Posts and Telecommunications. Her research interests include cryptography, information security and network security. E-mail: zhanghua_288@bupt.edu.cn

Qiaoyan Wen Qiaoyan Wen received the B.S. and M.S. degrees in Mathematics from Shaanxi normal University, Xi’an, China, in 1981 and 1984, respectively, and the Ph.D degree in cryptography from Xidian University, Xi’an, China, in 1997. She is a professor of Beijing University of Posts and Telecommunications. Her present research interests include coding theory, cryptography, information security, internet security and applied mathematics. E-mail: wqy@bupt.edu.cn

Kaitai Liang Kaitai Liang received the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, in 2014. He is currently an Assistant Professor with the Department of Computer Science, University of Surrey, U.K. His research interests are applied cryptography and information security, in particular, encryption, network security, big data security, privacy enhancing technology, blockchain, lattice-based crypto and security in cloud computing. E-mail: k.liang@surrey.ac.uk

Wei Yin, received the B.S. degree in Mathematics and Applied Mathematics from Huaibei Normal University, Huaibei, Anhui, China, in 2012, and the Ph.D degree in cryptography from Beijing University of Posts and Telecommunications, Beijing, China, in 2019. His research interests include public key cryptography, lattice cryptography, and provable security.