Hindawi
Security and Communication Networks
Volume 2021, Article ID 7034944, 16 pages
https://doi.org/10.1155/2021/7034944

Research Article

Tree-Based Public Key Encryption with Conjunctive Keyword Search

Yu Zhang,[1] **Lei You**,[1] **and Yin Li**[2]

[1]*School of Computer and Information Technology, Xinyang Normal University, Xinyang, China*
[2]*School of Cyberspace Security, Dongguan University of Technology, Dongguan, China*

Correspondence should be addressed to Lei You; leiyouxy@vip.163.com

Received 26 April 2021; Accepted 16 October 2021; Published 5 November 2021

Academic Editor: Hu Xiong

Searchable public key encryption supporting conjunctive keyword search is an important technique in today's cloud environment. Nowadays, previous schemes usually take advantage of forward index structure, which leads to a linear search complexity. In order to obtain better search efficiency, in this paper, we utilize a tree index structure instead of forward index to realize such schemes. To achieve the goal, we first give a set of keyword conversion methods that can convert the index and query keywords into a group of vectors and then present a novel algorithm for building index tree based on these vectors. Finally, by combining an efficient predicate encryption scheme to encrypt the index tree, a tree-based public key encryption with conjunctive keyword search scheme is proposed. The proposed scheme is proven to be secure against chosen plaintext attacks and achieves a sublinear search complexity. Moreover, both theoretical analysis and experimental result show that the proposed scheme is efficient and feasible for practical applications.

1. Introduction

With the rapid development of cloud computing, many enterprises and individuals are willing to store their data over the cloud server for the sake of its efficiency and convenience. Although data outsourcing has brought great convenience to resource-constrained users, it inevitably raises data privacy and security concerns since users' data stored in the cloud server are generally in plaintext mode. If the cloud server is compromised by some malicious intruder, these data can be accessed by them with no obstacles. To overcome this problem, a straightforward approach is encrypting data before outsourcing them. Nevertheless, this trivial approach needs to download all encrypted data to the client and then perform a keyword search after decrypting these data. Because this method is inefficient, if the amount of data is relatively large, exploring a new technology which can retrieve encrypted data on the cloud server without decryption is of prime importance in practical application.

Searchable encryption (SE) [1] is a promising method, which offers a capacity for searching over encrypted data securely. In terms of different cryptography primitives, SE can be classified into two types, that is, searchable symmetric encryption (SSE) and searchable public key encryption (SPE). SSE is fit for the scenario of one data owner and one data user, while SPE can be applied in the scenario of many data owners and one data user, e.g., Mail Routing Service (MRS) and Personal Healthcare Record (PHR). In practice, since user's data and query can be expressed as a keyword set, many existing SPE schemes are keyword-based schemes.

The first SPE scheme that supports a keyword search, called public key encryption with keyword search (PEKS), was proposed by Boneh et al. [2]. They first defined the framework and security definition of SPE and gave a concrete scheme supporting only a single keyword search. To support conjunctive keywords search, Park et al. gave two public key encryptions with conjunctive keyword search (PECK) schemes [3]. The first scheme requires too much bilinear pairing operations, while the second one requires many private keys that are linear with the number of keywords. However, their schemes need the keyword field as additional information to perform the keyword search. To

create a PECK scheme without keyword field, a hidden vector encryption scheme was presented by Boneh and Waters [4]. To improve the search efficiency, Zhang et al. proposed a keyword field-free PECK scheme with less search overhead [5]. In the years that followed, many works aim to improve the efficiency of PECK [6, 7], which enhances the practicability of PECK. Nevertheless, the search complexities of the aforementioned schemes are all linear with the number of keywords in a corpus, which is not practical when the corpus is large.

Motivation and Objective. To better clarify the motivation of our scheme, we give Table 1 to show some recent works that are relevant to our work. According to Table 1, we find whether most SPE schemes [5–12] own additional abilities or do not utilize forward index structure, which suffers from linear search complexity. In this case, these schemes are impractical in reality applications, as the volume of data increases. To improve the search efficiency, recently, a PECK scheme using a hidden structure was proposed, which can achieve a sublinear search efficiency [13]. However, the time consumption of index building is proportional to the square of the number of keywords in each document. For the efficiency concern, it is necessary to construct a PECK scheme with high efficiency in the aspects of encryption and search.

In the field of information retrieval, index structures with sublinear search complexity, such as inverted and tree-based index structures, are very popular and have already been adopted in the domain of SSE [14–18]. Considering that the tree structure commonly owns a better search performance than the forward index structure, thus, this paper aims to construct a PECK scheme under a tree index structure to achieve a sublinear search efficiency without sacrificing encryption efficiency. For simplicity, we name this scheme as tree-based public encryption with conjunctive keyword search (T-PECK).

Contributions. The contributions of our scheme are listed as follows:

(1) We design a set of keyword conversion methods, which are used to transform the keyword set in each document and a query into a group of vector representations. The semantic information embedded in these vector representations can be utilized to appraise the relationship between documents and queries.

(2) Based on the above conversion methods, a tree building algorithm is proposed to create a tree index for all documents. Correspondingly, a tree search algorithm is also given to realize conjunctive keyword search with a sublinear search complexity.

(3) Inspired by the definition of PECK proposed in [3], we give a detailed framework and security definition of T-PECK. Based on the framework, by taking advantage of an efficient predicate-only encryption supporting inner product scheme (PO-IPE) scheme introduced in [19] to encrypt the index tree, a concrete T-PECK scheme is proposed. Moreover, a rigorous proof is given to demonstrate the security of T-PECK according to the security definition.

In addition, to show the feasibility of our scheme, we also give a detailed experiment to verify the efficiency of our scheme. The experiment results given in Section 5 show that the search complexity of T-PECK is indeed sublinear with the number of the documents without sacrificing the encrypting efficiency, and T-PECK is more practical than recent PECK schemes [8, 13].

Related work. Generally speaking, SE can be categorized as SSE and SPE, where SSE and SPE are based on public key and symmetric key encryption systems, respectively. Although SPE is less efficient in terms of encryption and search than SSE as the use of public key cryptography, it allows more extensive application scenarios and better security properties.

The first SPE encryption solution supporting keyword search (PEKS), reckoned as a generalization of anonymous identity-based encryption (IBE) [20], was designed by Boneh et al. [2]. After this, Abdalla et al. gave definitions of computational and statistical consistency of PEKS and proposed a new PEKS scheme [21]. Considering that works in [2, 21] only realize a single keyword search, Park et al. first gave the first PECK scheme [3]. In their scheme, each keyword is affiliated with a keyword field. If two same keywords are associated with two different keyword field, they are reckoned as two different keywords. This property has limited it in applications of many scenarios. To achieve PECK without the keyword field, a hidden vector encryption (HVE) scheme was proposed by Boneh and Waters [4]. To support disjunctive keyword search, Katz et al. proposed a predicate encryption (PE) scheme supporting inner product [22], which is also called inner product encryption (IPE). In their scheme, the keyword sets of a document and a query can be converted to be an attribute and a predicate vector, respectively. By applying the PE scheme to encrypt these vectors, a PECK or a public key encryption with disjunctive keyword search (PEDK) can be obtained. In order to improve the query efficiency, some efficient PECK schemes [5, 6, 8] were proposed by reducing the bilinear pairing operations in the search process. To support conjunctive and disjunctive keyword search simultaneously, by adopting an efficient IPE scheme [23], Zhang et al. proposed a public key encryption with conjunctive and disjunctive keyword search (PECDK) scheme [24]. Then, they also presented a new PECDK scheme [25] which is more efficient than their previous scheme [24]. To support more expressive queries, Yang et al. devised a SPE scheme supporting various search functions, e.g., range search, conjunctive keyword search, and Boolean keyword search [7]. Although this scheme can support sophisticated query, the search efficiency of this scheme remains to be further improved. For efficiency concern, two SPE scheme [13, 26] supporting Boolean keyword search were proposed to improve the search efficiency. In addition, in recent years, some works are interested in other aspects of SPE, such as keyword guessing attack (KGA) [27–29], access control [30, 31], and fast search [32, 33]. These works profoundly improves the usability of SPE.

Table 1: Comparison between previous searchable encryption schemes and ours.

Ref.	Search function	Index structure	Setting	Additional special abilities
Ours	Conjunctive keyword search	Tree-based index	Public key	—
[5]	Conjunctive keyword search	Forward index	Public key	—
[6]	Conjunctive keyword search	Forward index	Public key	—
[7]	Boolean keyword search	Forward index	Public key	—
[8]	Conjunctive keyword search	Forward index	Public key	—
[9]	Conjunctive keyword search	Forward index	Public key	Verification for query results
[10]	Conjunctive keyword search	Forward index	Public key	Resistance for KGA
[11]	Single keyword search	Forward index	Public key	Access control
[12]	Single keyword search	Forward index	Public key	Certificateless encryption
[13]	Conjunctive keyword search	Hidden structure	Public key	—
[14]	Rank search	Tree-based index	Symmetric key	—
[15]	Rank search	Tree-based index	Symmetric key	—
[16]	Rank search	Tree-based index	Symmetric key	—
[17]	Rank search	Inverted index	Symmetric key	—
[18]	Rank search	Inverted index	Symmetric key	—

The additional special ability means that this scheme owns some extra security mechanism. KGA is the abbreviation of keyword guessing attack. Rank search is that the search result will be ranked before returning to the data users.

In the domain of SSE, the tree and inverted index structures are usually adopted in SSE schemes to achieve better search efficiency. The SSE schemes using inverted index were given in [17, 18]. By taking advantage of r-tree, kd-tree, and balanced binary tree, SSE schemes with the tree index structure were released in [14–16]. To the best of our knowledge, there is no PECK scheme based on the tree index structure so far. Thus, in this paper, to improve the search efficiency substantially, we attempt to construct a PECK scheme based on the balanced binary tree.

Organization. We organize this paper as follows. In Section 2, we present the framework of T-PECK and the security definition of T-PECK and introduce some backgrounds related to our scheme. The keyword conversion methods and the tree building algorithm are presented in Section 3. We construct T-PECK in Section 4, and a detailed security proof of T-PECK is also presented in this section. In Section 5, we give the theoretical and experimental analysis of T-PECK. Section 6 concludes our work.

2. Preliminary

In this section, we describe the application scenario and give the system model of the T-PECK scheme. Based on this system model, we present a formal framework and security definition of T-PECK. Besides, because PO-IPE scheme is adopted as the basic encryption module, we also introduce the bilinear pairing group and the definition of PO-IPE. To formulate our work mathematically, Table 2 shows notations that are adopted in this paper.

2.1. The Proposed T-PECK Model. In T-PECK scheme, there are three roles: data owners (DOs), data user (DU), and cloud server (CS). DU first generates a pair of keys for which the public key (pk) is open to the public and the secret key (sk) can be only obtained by DU. When one DO wants to send a group of documents to DU, he/she will encrypt these documents and send the encrypted documents along with an encrypted index tree. Note that this encrypted index tree is generated by using a group of keyword set $W_1, W_2, \ldots, W_d$ derived from these documents and pk. When DU wants to retrieve documents containing a specific list of keywords, he/she makes use of sk and query keywords to construct a trapdoor and sends the trapdoor to CS. Once the trapdoor is received, CS tests the encrypted index against the trapdoor and returns the matched documents to DU. The system model is illustrated in Figure 1. As shown in Figure 1, DO builds the secure index and sends it to the CS. When CS receives the trapdoor generated by DU, it performs the search algorithm and returns the query results to DO. According to this model, the definition of the framework of T-PECK is given as follows.

Definition 1. The T-PECK scheme consists of five probabilistic polynomial time (PPT) algorithms (**KeyGen**, **CreateTree**, **IndexBuild**, **Trapdoor**, and **Search**):

(1) **KeyGen** (γ): given a security parameter γ, this algorithm generates the public key pk and the secret key sk

(2) **CreateTree** (W): given a dataset $W = \{W_1, W_2, \ldots, W_d\}$ in which W_i is the corresponding keyword set of document f_i and $i \in [1, d]$, the algorithm outputs a plaintext tree T

(3) **IndexBuild** (pk, T): this algorithm builds a searchable encrypted index tree I_T by utilizing the public key pk and a plaintext tree T

(4) **Trapdoor** (sk, Q): given the secret key sk and a conjunctive keyword query $Q = \{q_1, q_2, \ldots, q_m\}$, where q_i is a query keyword and $i \in [1, m]$, this algorithm produces a trapdoor T_Q

(5) **Search** (pk, T_Q, I_T): this algorithm takes pk, T_Q, and I_T as input and returns a group of documents by searching the index tree

Correctness property. For a conjunctive keyword query Q and a group of keyword sets $W = \{W_1, W_2, \ldots, W_d\}$, for

Table 2: Notation.

$F = \{f_1, f_2, \ldots, f_d\}$	All documents in a corpus
$C = \{c_1, c_2, \ldots, c_d\}$	Encrypted documents of F
$\text{Dic} = \{w_1, w_2, \ldots, w_N\}$	All keywords in F (dictionary for F)
T	The plaintext index tree for F
I_T	The encrypted index tree for T
$W_i = \{w_{i1}, w_{i2}, \ldots, w_{\text{in}}\}$	The keyword set of f_i
$Q = \{q_1, q_2, \ldots, q_m\}$	The keyword set of query Q
T_Q	The trapdoor of Q
n	The number of keywords in each document W_i
m	The number of keywords in the query Q
$\overrightarrow{W_i}$	The vector representation for W_i
$\{S_1, S_2, \ldots, S_l\}$	A group of document sets obtained by splitting Dic
q_i	A keyword in Q, $i \in [1, m]$
w_{ij}	A keyword in W_i, $i \in [1, d]$ and $j \in [1, n]$

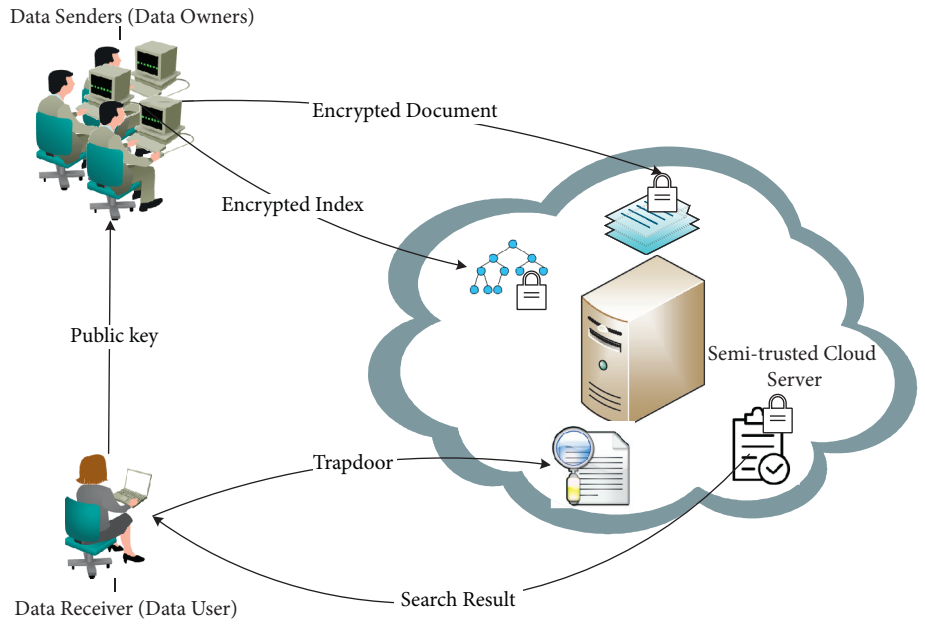

Figure 1: Architecture of the search over encrypted cloud data.

correctly generating $\text{KeyGen}(\gamma) \longrightarrow \{\text{pk}, \text{sk}\}$, **CreateTree**$(W) \longrightarrow T$, $\text{IndexBuild}(\text{pk}, T) \longrightarrow I_T$, and $\text{Trapdoor}(sk, Q) \longrightarrow T_Q$, if $Q \subseteq W_i$, the algorithm **Search**(pk, T_Q, I_T) outputs the document f_i. Otherwise, it outputs f_i with negligible probability.

Actually, in the T-PECK scheme, the index tree T and the document set F are encrypted by $\text{IndexBuild}(\text{pk}, T)$ and $\text{Enc}(\text{key}, F)$, respectively, where $\text{Enc}(\cdot)$ is a secure symmetric encryption scheme, e.g., AES-CBC. Moreover, in practice, the Search algorithm only returns the identifiers of matched documents, and then, DU will make use of these identifiers to look up the corresponding ciphertexts. Thus, like other related work, our work only concentrates on searchable encryption part.

2.2. Security Definition of T-PECK. Before presenting the formal security definition of T-PECK, based on four kinds of private leaks, we first define a leakage function under four patterns introduced in [34], which is inevitably exposed to the cloud server.

Size Pattern. The size of encrypted documents and queries can be accessed by the cloud server as they are stored in the cloud. This leakage is denoted as the leak of size pattern.

Access Pattern. The cloud server can obtain the relationship between a specific query and identifiers of matched documents, which is called the leakage of access pattern.

Search Pattern. Given a document set $F = \{f_1, f_2, \ldots, f_d\}$ and a query set $Q = \{Q_1, Q_2, \ldots, Q_t\}$, a $d \times t$ matrix can be constructed by the cloud server. In this matrix, if f_i matches the query Q_j, the element in the ith row and jth column is set to be 1. The search pattern is the information of the matrix.

Path Pattern. For a specific query, when the search algorithm traverses in the tree, the path will be revealed to the cloud

server. This path information can be regarded as the leakage of the path pattern.

These leakage patterns are default in most searchable encryption schemes [5–8]. Actually, the technique called oblivious RAMs can be adopted to protect the access pattern and the search pattern, but it suffers from the problem of low efficiency in the actual applications. Thus, we do not consider these leakages in this paper.

Similar to the definition presented in [3], for two challenge index trees $T^{(0)}$ and $T^{(1)}$ based on two datasets $W^{(0)}$ and $W^{(1)}$, we propose a security definition which demands that the encrypted index tree of $T^{(0)}$ and that of $T^{(1)}$ are computationally indistinguishable for any probabilistic polynomial time adversary A. Given a leakage function $L(T, Q)$ in which T is an index tree of a dataset D and Q is a keyword query, the detailed security definition is given as follows.

Definition 2. For any probabilistic polynomial time (PPT) adversary $\mathbb{A}$, using a security parameter γ, A T-PECK scheme can be seen as semantical security against chosen plaintext attacks if $\mathbb{A}$'s advantage is negligible under the following game.

(1) Setup: the challenger $\mathbb{C}$ utilizes KeyGen (γ) to create pk and sk and sends pk to $\mathbb{A}$.

(2) Phase 1: for any query Q that $\mathbb{A}$ wants to perform, $\mathbb{A}$ adaptively ask $\mathbb{C}$ for the trapdoor T_Q of Q.

(3) Challenge: let $Q^{(1)}, Q^{(2)}, \ldots, Q^{(t)}$ be t keyword queries that are performed in Phase 1. Two challenge index trees $T^{(0)}$ and $T^{(1)}$ are chosen and sent to $\mathbb{C}$ under the restriction that $L(T^{(0)}, Q^{(i)}) = L(T^{(1)}, Q^{(i)})$ for each $i \in [1, t]$. After this, $\mathbb{C}$ selects a random bit $\beta \in \{0, 1\}$, builds an encrypted index tree $I_T^{(\beta)}$ by using IndexBuild$(\text{pk}, T^{(\beta)})$, and sends $\{I_T^{(\beta)}, T^{(0)}, T^{(1)}\}$ to $\mathbb{A}$.

(4) Phase 2: for any keyword query Q, $\mathbb{A}$ can continue to request the trapdoor T_Q under the restriction $L(T^{(0)}, T_Q) = L(T^{(1)}, T_Q)$.

(5) Response: $\mathbb{A}$ outputs $\beta\prime \in \{0, 1\}$. If $\beta' = \beta$, $\mathbb{A}$ wins the game.

According the above game, the advantage of $\mathbb{A}$ is defined as

$$\text{Adv}_{\text{Game}}^{\mathbb{A}} = \left| \Pr[\beta' = \beta] - \frac{1}{2} \right|. \quad (1)$$

2.3. The Concept of PO-IPE Scheme

2.3.1. Framework of PO-IPE. The original definition of PO-IPE was presented in [19]. In this definition, a ciphertext is associated with an attribute vector $\vec{x}$, and a key is associated with a predicate vector $\vec{v}$ correspondingly. The significant property of PO-IPE is that a ciphertext of $\vec{x}$ can be decrypted by a key of $\vec{v}$ if and only if $\vec{v} \cdot \vec{x} = 0$. More explicitly, the framework of PO-IPE is introduced in the following definition.

Definition 3. $\sum$ and $\mathbb{F}$ are represented as an arbitrary set of attributes and predicated on $\sum$, respectively. There are four PPT algorithms in a PO-IPE scheme with $\sum$ and $\mathbb{F}$, i.e.,

(1) **Setup**: this algorithm inputs a security parameter γ and outputs pk and master secret key (msk)

(2) **KeyGen**: given msk and a predicate vector $\vec{v} \in \mathbb{F}$ as input, this algorithm outputs the secret key $sk_{\vec{v}}$ of $\vec{v}$

(3) **Enc**: this encryption algorithm uses pk to encrypt an attribute vector $\vec{x} \in \sum$ and outputs the ciphertext C

(4) **Dec**. taking pk, $sk_{\vec{v}}$, and C as input, this decryption algorithm outputs 1 or 0 either

Consistency of PO-IPE. For correctly generating **Setup**$(1^n) \longrightarrow \{\text{pk}, \text{msk}\}$, KeyGen$(\text{msk}, \vec{v}) \longrightarrow sk_{\vec{v}}$, and Enc$(\text{pk}, \vec{x}) \longrightarrow C$, for all $\vec{v} \in \mathbb{F}$ and $\vec{x} \in \Sigma$, the decryption algorithm **Dec**$(\text{pk}, sk_{\vec{v}}, C)$ outputs 1 if $\vec{v} \cdot \vec{x} = 0$, otherwise 0.

According to the definitions of T-PECK and PO-IPE, by converting the index and query keyword sets into an attribute vector $\vec{x}$ and a predicate vector $\vec{v}$, respectively, we can take advantage of a PO-IPE scheme to create our scheme. The key to realizing the goal of conjunctive keyword search is constructing a keyword conversion method to change the query and index keyword sets into vectors.

2.3.2. Security Definition of PO-IPE. The security of T-PECK depends on that of PO-IPE since PO-IPE is the basis of T-PECK. To present a detailed proof for the security of T-PECK, the security definition of PO-IPE given in [19] is described as follows.

Definition 4. Given a security parameter γ, for any PPT adversary $\mathbb{A}$, if $\mathbb{A}$'s advantage is negligible in the game described below, we say that a PO-IPE scheme is semantically secure against chosen plaintext attacks.

(1) Setup: $\mathbb{C}$ creates pk and msk by utilizing Setup (γ) and sends pk to $\mathbb{A}$.

(2) Phase 1: $\mathbb{A}$ can ask $\mathbb{C}$ for t predicate vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_t$ adaptively and obtain corresponding keys $sk_{\vec{v}_1}, sk_{\vec{v}_2}, \ldots, sk_{\vec{v}_t}$.

(3) Challenge: two challenge attribute vectors $\vec{x}^{(0)}, \vec{x}^{(1)}$ are selected randomly by $\mathbb{A}$ under the following constraints. For secret keys $sk_{\vec{v}_1}, sk_{\vec{v}_2}, \ldots, sk_{\vec{v}_t}$ requested in Phase 1, one of the following constraints should be satisfied.

$\vec{v}_j \cdot \vec{x}^{(0)} \neq 0$ and $\vec{v}_j \cdot \vec{x}^{(1)} \neq 0$.
$\vec{v}_j \cdot \vec{x}^{(0)} = 0$ and $\vec{v}_j \cdot \vec{x}^{(1)} = 0$, where $j \in [1, t]$.
$\mathbb{C}$ randomly chooses a bit β and sends the ciphertext $C^{(\beta)} \longrightarrow \text{Enc}(\text{pk}, \vec{x}^{(\beta)})$ to $\mathbb{A}$.

(4) Phase 2: $\mathbb{A}$ can continue to request keys $sk_{\vec{v}_{t+1}}, sk_{\vec{v}_{t+2}}, \ldots, sk_{\vec{v}_q}$, for other predicate vectors, $\vec{v}_{t+1}, \vec{v}_{t+2}, \ldots, \vec{v}_q$. These keys must comply with the restriction given in the challenge phase.

(5) Response: a bit $\beta\prime$ is given by $\mathbb{A}$. If $\beta\prime = \beta$, $\mathbb{A}$ wins the game.

$\mathbb{A}$'s advantage under the above game is described as

$$\mathrm{Adv}_{\mathrm{Game}}^{\mathbb{A}} = \left|\Pr[\beta\prime = \beta] - \frac{1}{2}\right|. \quad (2)$$

3. Construction of Index Tree

In this section, we introduce three methods related to building the index tree. The first method is a keyword conversion method which changes document and query keywords into an attribute vector and a predicate vector, respectively. This conversion method is used to construct leaf nodes. The second method aims to create "0-1" vectors of internal nodes of the index tree. Through randomly partitioning all the keywords in the corpus into a set of clusters, we devise a "converting and merging" algorithm to create "0-1" vectors based on these clusters. The third method is a recursive algorithm, which elaborately combines the first and second methods to create the index tree.

3.1. Keywords' Conversion Method. We assume that any keyword w can be expressed as $\{0,1\}^*$ and define a function H_1: $\{0,1\}^* \longrightarrow Z_p^*$. H_1 is collision resistance since p is a prime larger than the number of words in a dictionary. That is to say, if $i \neq j$, then $H_1(w_i) \neq H_1(w_j)$, where w_i and w_j are two different keywords. For a document keyword set $W = \{w_1, w_2, \ldots, w_n\}$ and a query $Q = \{q_1, q_2, \ldots, q_m\}$, the approach that converts W and Q into an attribute vector and a predicate vector, respectively, is described as follows:

(1) For the keyword set $W = \{w_1, w_2, \ldots, w_n\}$, we construct a function:

$$f(x) = (x - H_1(w_1))(x - H_1(w_2)) \ldots (x - H_1(w_n)) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0 x^0. \quad (3)$$

Based on coefficients of $f(x)$, the vector representation $\vec{a} = \{a_0, a_1, \ldots, a_n\}$ of W is now developed.

(2) Similarly, for a query $Q = \{q_1, q_2, \ldots, q_m\}$, we can construct a vector $\vec{x} = \{x_0, x_1, \ldots, x_n\}$, where $x_i = H_1(q_1)^i + H_1(q_2)^i + \cdots + H_1(q_m)^i$ and $i \in [0, n]$. $\vec{x}$ is the vector representation of Q.

(3) It is noteworthy that if there is $Q \subseteq W$, we can verify that $\vec{a} \cdot \vec{x} = 0$. This property can be used to test whether Q is a subset of W.

As a result, we can convert the keyword set of each document into an attribute vector $\vec{a}$ and transform each query into a predicate vector $\vec{x}$ by adopting the keyword conversion method. These vectors will be used to create the leaf nodes.

3.2. Construction of "0-1" Vectors. Suppose that all the keywords in a corpus is $\mathrm{DIC} = \{w_1, w_2, \ldots, w_N\}$; we randomly split DIC into a group of document sets $\{S_1, S_2, \ldots, S_l\}$, where $\cup_{i=1}^{l} S_i = \mathrm{DIC}$, $S_i \cap S_j = \varnothing$, $|S_i| \approx |S_j|$, and $i \neq j$. For document sets, $S = \{S_1, S_2, \ldots, S_l\}$, we give the method of how to use S to construct a vector representing for an internal node or a query as follows.

For each document along with a keyword set, $W = \{w_1, w_2, \ldots, w_n\}$, the algorithm for converting a document into a "0-1" vector is described in Algorithm 1. As shown in Algorithm 1, given a document, we first create a zero vector $\vec{v}_{01}$ whose length is $l + 1$. Then, for each keyword, $w_i \in W$, if $w_i \in S_j$, the value of the jth dimension of $\vec{v}_{01}$ is set to be 1, where $i \in [1, n]$ and $j \in [1, l]$. Finally, the value of the last dimension is set to be 1. For example, if $n = 3$ and $l = 5$, suppose that $w_1 \in S_2$, $w_2 \in S_4$, and $w_3 \in S_5$, and the vector $\vec{v}_{01}$ of $W = \{w_1, w_2, w_3\}$ is $\{0, 1, 0, 1, 1, 1\}$.

For a query Q along with a keyword set, $Q = \{q_1, q_2, \ldots, q_m\}$, the algorithm for converting a query into a "0-1" vector is described in Algorithm 2. As shown in Algorithm 2, for the query Q, we first create a zero vector $\vec{q}_{01}$ whose length is $l + 1$. Then, for each keyword $q_i \in Q$, if $q_i \in S_j$, the value of the jth dimension of $\vec{q}_{01}$ is increased by one, where $i \in [1, m]$ and $j \in [1, l]$. At last, the last dimension's value is set to be $-1 \times m$. For example, if $m = 2$ and $l = 5$, suppose that $q_1 \in S_1$ and $q_2 \in S_5$, the vector $\vec{q}_{01}$ of $Q = \{q_1, q_2\}$ is $\{1, 0, 0, 0, 1, -2\}$. It can be verified that $\vec{v}_{01} \cdot \vec{q}_{01} = 0$ if $Q \subseteq W$.

Moreover, taking two "0-1" vectors as input, we also give a merging algorithm to generate a new "0-1" vector, which is very useful for creating an internal node. The merging algorithm is described in Algorithm 3. Suppose that $\vec{v}$ is initialized to be a zero vector; the essence of Algorithm 3 is setting $\vec{v}[i] = 1$ if either $\vec{v}_1[i]$ or $\vec{v}_2[i]$ equals 1, where $i \in [1, l + 1]$. For example, if $\vec{v}_1 = \{0, 1, 0, 1, 0, 1\}$ and $\vec{v}_2 = \{1, 1, 0, 0, 0, 1\}$, the output of Algorithm 3 is $\vec{v} = \{1, 1, 0, 1, 0, 1\}$.

Note that, the "0-1" vector for each document is only used in "CreateTree" algorithm and discarded when the "CreateTree" algorithm is finished.

3.3. Tree Building Algorithm. The tree building algorithm mainly consists of two steps. The first one is to create the leaf node for each document. Each leaf node has two vectors: one is an attribute vector obtained by using keyword conversion method; the other is a "0-1" vector generated by using Algorithm 1. The second step is to create the internal node in a "bottom-up" manner. The internal node contains one "0-1" vector generated by using Algorithm 3. The detailed algorithms are presented in Algorithm 4 and 5.

Algorithm 5 is for building an index tree and declared by BuildTree (CurrentNodeSet).

A formal definition of the data structure of a tree node u is $u = \langle \mathrm{ID}, \vec{u}_a, \vec{u}_{01}, P_l, P_r, \mathrm{FID} \rangle$, where ID is u's identity, $\vec{u}_a$ and $\vec{u}_{01}$ are the representation vectors of u, P_l and P_r are two pointers that point u's left and right children, respectively, and FID is the identity of a document when u is a leaf node. Let GenID be a function that can generate a unique ID for each node. According to Algorithm 4, for the leaf node

Input: a keyword set $W = \{w_1, w_2, \ldots, w_n\}$ of a document and a group of keyword sets $S = \{S_1, S_2, \ldots, S_l\}$.
Output: a "0-1" vector for the document.
(1) Creates a zero vector $\overrightarrow{v}_{01}$ whose length is $l + 1$.
(2) **for** $i = 1; i <= n; i++$ **do**
(3) **for** $j = 1; j <= l; j++$ **do**
(4) **if** $w_i \in S_j$ **then**
(5) $\overrightarrow{v}_{01}[j] = 1;$
(6) **end if**
(7) **end for**
(8) **end for**
(9) $\overrightarrow{v}_{01}[l + 1] = 1;$
(10) **return** $\overrightarrow{v}_{01};$

Algorithm 1: Converting a document into a "0-1" vector.

Input: a query $Q = \{q_1, q_2, \ldots, q_m\}$ and a group of keyword sets $S = \{S_1, S_2, \ldots, S_l\}$.
Output: a "0-1" vector for the query.
(1) Creates a zero vector $\overrightarrow{q}_{01}$ whose length is $l + 1$.
(2) **for** $i = 1; i <= m; i++$ **do**
(3) **for** $j = 1; j <= l; j++$ **do**
(4) **if** $q_i \in S_j$ **then**
(5) $\overrightarrow{q}_{01}[j] += 1;$
(6) **end if**
(7) **end for**
(8) **end for**
(9) $\overrightarrow{q}_{01}[l + 1] = -1 \times m;$
(10) **return** $\overrightarrow{q}_{01};$

Algorithm 2: Converting a query into a "0-1" vector.

Input: two "0-1" vectors $\overrightarrow{v}_1$ and $\overrightarrow{v}_2$
Output: a new "0-1" vector obtained by merging $\overrightarrow{v}_1$ and $\overrightarrow{v}_2$.
(1) Creates a zero vector $\overrightarrow{v}$ whose length is $l + 1$.
(2) **for** $i = 1; i <= l; i++$ **do**
(3) **if** $\overrightarrow{v}_1[i] == 1$ or $\overrightarrow{v}_2[i] == 1$ **then**
(4) $\overrightarrow{v}[i] = 1$
(5) **end if**
(6) **end for**
(7) $\overrightarrow{v}[l + 1] = 1;$
(8) **return** $\overrightarrow{v}$

Algorithm 3: A merging algorithm for two "0-1" vectors.

associated with the document f_i, the algorithm runs GenID to obtain an unique ID for the node and sets the value of FID as the identifier of f_i. Since there is no child for the leaf node, both P_l and P_r are set to be NULL. Based on the keyword set of f_i, $\overrightarrow{u}_a$ and $\overrightarrow{u}_{01}$ are generated by utilizing the keyword converting algorithm and Algorithm 1, respectively.

Based on all leaf nodes generated by Algorithm 4, the tree building algorithm is presented in Algorithm 5. CurrentNodeSet involves a group of nodes without a parent node. Let |CurrentNodeSet| be the number of nodes in CurrentNodeSet. If |CurrentNodeSet| equals 1, this means that the sole node in CurrentNodeSet is the root of the index tree. Otherwise, the internal nodes should be generated by using each pair of nodes in CurrentNodeSet. Concretely, TempNodeSet is initialized as an empty set which is used to store the newly generated nodes. Suppose that CurrentNodeSet$[i]$ and CurrentNodeSet$[i + 1]$ are a pair of nodes in CurrentNodeSet; a parent node u of these two nodes is created as follows. ID of u is generated by using the function GenID. P_l and P_r point to the nodes CurrentNodeSe$[i]$ and CurrentNodeSet$[i + 1]$, respectively. Based on the "0-1" vectors of CurrentNodeSe$[i]$ and CurrentNodeSet$[i + 1]$, $\overrightarrow{u}_{01}$ is generated by utilizing Algorithm 3. Both FID and $\overrightarrow{u}_a$ are set to be NULL since u is an internal node. Once the node u is created, it will be added to TempNodeSet. Note that if |CurrentNodeSet| is not even,

Input: a group of keyword sets $\{W_1, W_2, \ldots, W_d\}$, where W_i is the keyword set of f_i and $i \in [1, d]$.
Output: A group of leaf nodes.
(1) **for** each $i \in [1, d]$ **do**
(2) Constructs a leaf node u of f_i, sets u.ID = GenID, $u.P_l = u.P_r$ = NULL, u.FID be the identifier of f_i, and generates $\overrightarrow{u}_a$ and $\overrightarrow{u}_{01}$ according to the keywords conversion method and Algorithm 1, respectively;
(3) Inserts u to CurrentNodeSet;
(4) **end for**
(5) **return** *CurrentNodeSet*

Algorithm 4: Creating leaf nodes.

Input: the CurrentNodeSet, which contains all the leaf nodes.
Output: the plaintext index tree T.
(1) **if** |CurrentNodeSet| == 1 **then**
(2) **return** CurrentNodeSet;
(3) \\ This is the root node of the tree.
(4) **end if**
(5) Initializes an empty set TempNodeSet;
(6) Sets k = |CurrentNodeSet| and $i = 0$;
(7) **while** $i < k$ **do**
(8) **if** $i + 1 < k$ **then**
(9) Creates a parent node u for two nodes CurrentNodeSet$[i]$ and CurrentNodeSet$[i + 1]$, u.ID = GenID, $u.P_l$ = CurrentNodeSet$[i]$, $u.P_r$ = CurrentNodeSet$[i + 1]$, u.FID = NULL, $\overrightarrow{u}_a$ = NULL and generates $\overrightarrow{u}_{01}$ according to the Algorithm 3;
(10) Insert u to TempNodeSet;
(11) **else**
(12) Insert CurrentNodeSe$[i]$ to TempNodeSet;
(13) **end if**
(14) $i = i + 2$;
(15) **end while**
(16) CurrentNodeSet = BuildTree(TempNodeSet);
(17) \\ recursive calls BuildTree (Algorithm 5).
(18) **return** CurrentNodeSet;

Algorithm 5: The algorithm for building an index tree and declared by BuildTree (CurrentNodeSet).

the last node in CurrentNodeSet is added to TempNodeSet straightforwardly. Through calling Algorithm 5 recursively, the plaintext index tree can be built.

Example 1. An example of building an index tree is illustrated in Figure 2. From Figure 2, an index tree of $F = \{f_1, f_2, \ldots, f_8\}$ is given, where the dimension of the "0-1" vector for each node is 6. There are three steps for building the tree. The first step is changing the keyword set of each document into an attribute vector (denoted by $\overrightarrow{a_i}$) and a "0-1" vector (denoted by a sequence of "0-1") by using the keyword conversion method and Algorithm 1, respectively. After this step, each document is transformed into a leaf node with an attribute vector and a "0-1" vector. The second step is building the index tree based on the "0-1"vectors of the leaf nodes in a bottom-up manner by taking advantage of Algorithm 4 and 5. After this step, the index tree has been established, and each internal node in the tree owns a "0-1" vector. The third step is deleting all "0-1" vectors in the leaf nodes. The reason why we delete these vectors is that the "0-1" vectors contained in leaf nodes are only used to construct internal nodes of the index tree. In the search process, the leaf node only utilizes the attribute vector to perform keywords matching test. This step can reduce the storage cost of the index tree.

4. The Proposed Scheme

In this section, we give a T-PECK scheme with sublinear search complexity based on the aforementioned algorithms. After this concrete construction, a detailed proof is given to demonstrate the security of T-PECK based on the security definition.

4.1. Construction of T-PECK. According to Definition 3, we set $\text{Setup}_{\text{IPE}}$, $\text{KeyGen}_{\text{IPE}}(pk_{\text{IPE}}, \text{msk}_{\text{IPE}}, \overrightarrow{v})$, $\text{Enc}_{\text{IPE}}(pk_{\text{IPE}}, \overrightarrow{x})$, and $\text{Dec}_{\text{IPE}}(pk_{\text{IPE}}, c, sk_{\overrightarrow{v}})$ as four algorithms in PO-IPE, where pk_{IPE} and msk_{IPE} are the public key and master secret key, respectively, $\overrightarrow{x}$ and $\overrightarrow{v}$ are the attribute vector and predicate vector, respectively, and c and $sk_{\overrightarrow{v}}$ are the ciphertext and secret key generated by utilizing Enc_{IPE} and $\text{KeyGen}_{\text{IPE}}$, respectively. According to the PO-IPE scheme [19], we construct our T-PECK scheme as follows.

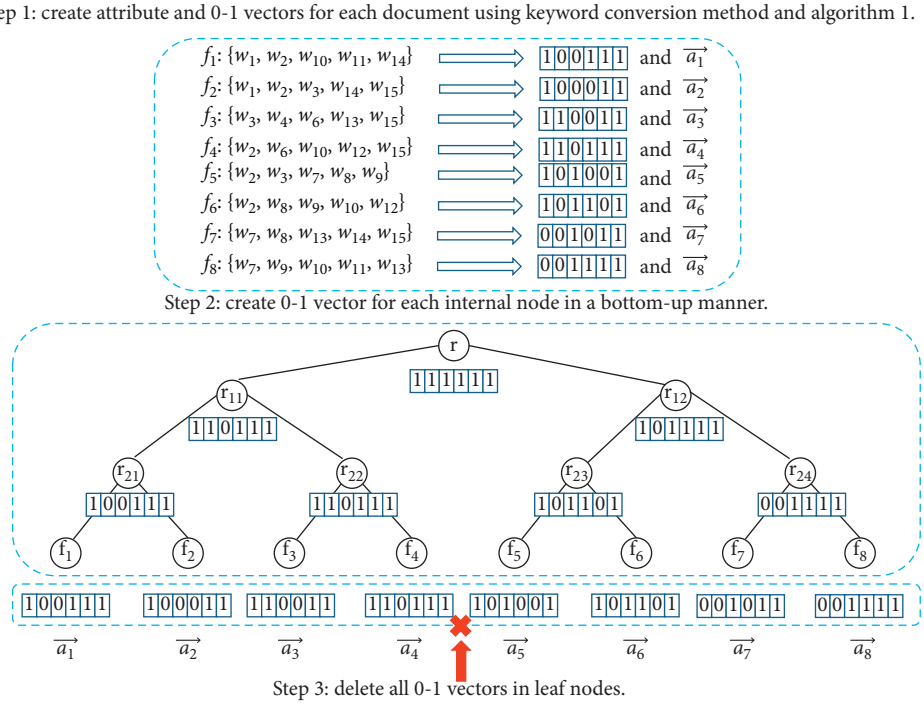

Figure 2: An example of building an index tree.

KeyGen (γ): given a security parameter γ, using the $\text{Setup}_{\text{IPE}}$ algorithm, this algorithm generates pk_{IPE} and msk_{IPE} and then sets $pk = pk_{\text{IPE}}$ and $sk = \text{msk}_{\text{IPE}}$ in which pk is given to data owner and sk is given to the data user.

CreateTree (W): given a group of keyword sets $\{W_1, W_2, \ldots, W_d\}$ in which each keyword set W_i is associated with a document f_i, this algorithm builds the index tree T according to Algorithm 4 and 5.

IndexBuild (pk, T): in order to encrypt the index tree T, a subalgorithm "encryptNode (pk, u)" is presented in Algorithm 6, where u is a tree node. As shown in Algorithm 6, if u is a leaf node of T, "encryptNode" calls $\text{Enc}_{\text{IPE}}(pk, \overrightarrow{u}_a)$ to generate $c_{\overrightarrow{u}_a}$ and replaces $\overrightarrow{u}_a$ with $c_{\overrightarrow{u}_a}$. Otherwise, it calls $\text{Enc}_{\text{IPE}}(pk, \overrightarrow{u}_{01})$ to generate $c_{\overrightarrow{u}_{01}}$ and replaces $\overrightarrow{u}_{01}$ with $c_{\overrightarrow{u}_{01}}$ and then recursively traverses its children nodes. Based on Algorithm 6, the IndexBuild algorithm calls encryptNode (pk, r) to generate the encrypted index I_T, where r is the root node of T, and uploads I_T to the cloud server.

Trapdoor (sk, pk, Q): given a keyword query $Q = \{q_1, q_2, \ldots q_m\}$, the algorithm first generates a predicate vector $\overrightarrow{v}$ and a "0-1" vector $\overrightarrow{q}_{01}$ by utilizing the keyword conversion approach and Algorithm 2, respectively. Then, it generates $t_1 = \text{KeyGen}_{\text{IPE}}(\text{sk}, \text{pk}, \overrightarrow{x})$ and $t_2 = \text{KeyGen}_{\text{IPE}}(\text{sk}, \text{pk}, \overrightarrow{q}_{01})$. Finally, it sends the trapdoor $T_Q = \{t_1, t_2\}$ of the query Q to the cloud server.

Search (pk, T_Q, I_T): in order to search the encrypted index tree I_T, a subalgorithm "searchNode (pk, T_Q, u, RList)" is given in Algorithm 7, where u is a tree node and RList is used to store the search result. Concretely, for an internal node u, "searchNode" computes $R = \text{Dec}_{\text{IPE}}(\text{pk}, c_{\overrightarrow{u}_{01}}, t_2)$. If $R = 1$, it continues to search the children nodes of u in a recursive way; Otherwise, it stops. For a leaf node u, the algorithm computes $R = \text{Dec}_{\text{IPE}}(\text{pk}, c_{\overrightarrow{u}_{01}}, t_1)$. If $R = 1$, it adds the FID of this node in RList. Based on Algorithm 7, the "Search" algorithm first initializes an empty set RList, then calls searchNode (pk, T_Q, r, and RList) to obtain all documents matched to the query Q, where r is the root node of T, and finally outputs RList to the data user.

Example 2. We also give Figure 3 to illustrate the search process. From Figure 3, we convert the query $Q = \{w_3, w_6, w_{15}\}$ into a "0-1" vector and a predicate vector $\overrightarrow{v}$ by using Algorithm 2 and the keyword conversion method, respectively. Based on the index tree given in example 1, the search process begins with the root node and first reaches to leaf nodes f_3 and f_4 since "0-1" vectors of r_{11} and r_{22} are both matching the "0-1" vector of the query Q. Then, the search algorithm computes values of $\overrightarrow{a}_3 \cdot \overrightarrow{v}$ and $\overrightarrow{a}_4 \cdot \overrightarrow{v}$ and adds f_3 to the result list RList due to $\overrightarrow{a}_3 \cdot \overrightarrow{v} = 0$ and $\overrightarrow{a}_4 \cdot \overrightarrow{v} \neq 0$. After that, the search algorithm checks the internal node r_{12}. Since the "0-1" vector of the query fails to match that of the node r_{12}, the children of r_{12} will not be reached. Finally, the algorithm outputs RList = $\{f_3\}$.

4.2. Security Proof. Our T-PECK scheme is constructed over the PO-IPE scheme. Thus, T-PECK's security is guaranteed by the fully secure PO-IPE scheme. To demonstrate the security of T-PECK, we give Proposition 1.

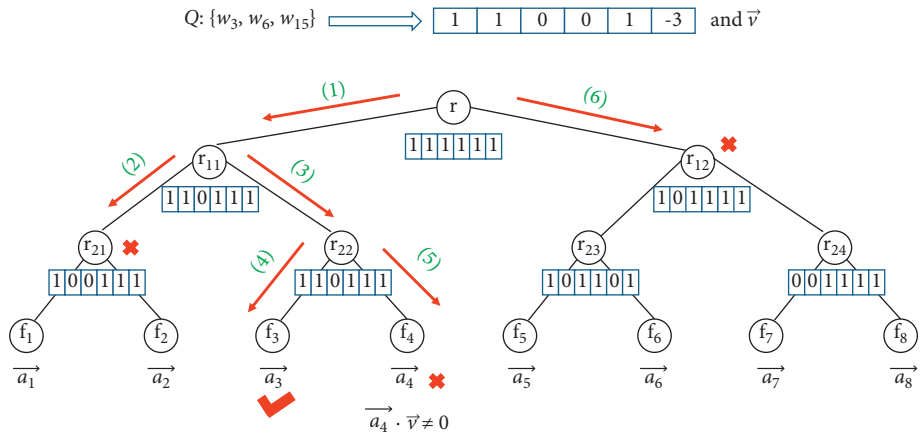

Figure 3: An example of search process.

Input: a tree node u and the public key pk.
Output: an encrypted node u.
(1) **if** u is NULL **then**
(2) **return**
(3) **end if**
(4) **if** u is a leaf node **then**
(5) calls $\mathrm{Enc}_{\mathrm{IPE}}(\mathrm{pk}, \vec{u}_a)$ to generate $c_{\vec{u}_a}$, and replaces $\vec{u}_a$ with $c_{\vec{u}_a}$;
(6) **else** \\ This means u is an internal node.
(7) calls $\mathrm{Enc}_{\mathrm{IPE}}(\mathrm{pk}, \vec{u}_{01})$ to generate $c_{\vec{u}_{01}}$, and replaces $\vec{u}_{01}$ with $c_{\vec{u}_{01}}$;
(8) calls encryptNode (pk, $u.P_l$);
(9) calls encryptNode (pk, $u.P_r$);
(10) **end if**
(11) **return**

Algorithm 6: The algorithm for encrypting each node in the index tree, declared by encryptNode (pk, u).

Input: the public key pk, a trapdoor T_Q for query Q, an encrypted node u, and a matched document set RList.
Output: RList.
(1) **if** u is NULL **then**
(2) **return**
(3) **end if**
(4) **if** u is a leaf node **then**
(5) computes $R = \mathrm{Dec}(\mathrm{pk}, c_{\vec{u}_a}, t_1)$;
(6) **if** $R == 1$ **then**
(7) adds u.FID to the RList;
(8) **end if**
(9) **else** \\ This means u is an internal node.
(10) calls $R = \mathrm{Dec}(\mathrm{pk}, c_{\vec{u}_{01}}, t_2)$;
(11) **if** $R == 1$ **then**
(12) searchNode (pk, T_Q, $u.P_l$, RList);
(13) searchNode (pk, T_Q, $u.P_r$, RList);
(14) **end if**
(15) **end if**
(16) **return**

Algorithm 7: The algorithm for searching each node in the index tree: searchNode (pk, T_Q, u, RList).

Proposition 1. *If the PO-IPE scheme is semantically secure against chosen plaintext attacks, our T-PECK scheme is L-semantically secure against chosen plaintext attacks, where L is the leakage function given in Section 2.2.*

Proof. We can argue that a PPT algorithm $\mathbb{A}$ can break the PO-IPE scheme if $\mathbb{A}$ can break the T-PECK scheme. The proof process is given as follows.

(1) Setup: the challenger $\mathbb{C}$ utilizes the $\text{Setup}_{\text{IPE}}$ algorithm to generate pk_{IPE} and msk_{IPE} and sets $\text{pk} = pk_{\text{IPE}}$ and $\text{sk} = \text{msk}_{\text{IPE}}$, where pk and sk are the public key and secret key of T-PECK.

(2) Phase 1: $\mathbb{A}$ can adaptively request trapdoors of queries $\{Q^{(1)}, Q^{(2)}, \ldots, Q^{(t)}\}$. For each query $Q^{(i)}$, the challenger $\mathbb{C}$ first generates two vectors $\overrightarrow{v}_i$ and $\overrightarrow{q}_{01}^{(i)}$ and then uses the $\text{KeyGen}_{\text{IPE}}$ algorithm to generate the trapdoor $T_{Q^{(i)}} = \{t_1^{(i)}, t_2^{(i)}\}$ in which $t_1^{(i)} = \text{KeyGen}_{\text{IPE}}(\text{pk}, \text{sk}, \overrightarrow{v}_i)$ and $t_2^{(i)} = \text{KeyGen}_{\text{IPE}}(\text{pk}, \text{sk}, \overrightarrow{q}_{01}^{(i)})$, where $i \in [1, t]$. Note that, each trapdoor can be seen as two decryption keys of PO-IPE.

(3) Challenge: after phase 1, $\mathbb{A}$ generates two plaintext trees T^0 and T^1 under the restriction that $L(T^{(0)}, Q^{(i)}) = L(T^{(1)}, Q^{(i)})$, where $i \in [1, t]$, and sends these two trees to $\mathbb{C}$. Note that, $\mathbb{C}$ can select two leaf nodes $u^{(0)}$ and $u^{(1)}$ from $T^{(0)}$ and $T^{(1)}$, respectively, as two challenge nodes of PO-IPE. After receiving them, flipping a coin $\beta \in \{0, 1\}$, $\mathbb{C}$ generates an index tree $I_{T^{(\beta)}}$ by running $\text{IndexBuild}(\text{pk}, T^{(\beta)})$ and sends it to $\mathbb{A}$. According to the IndexBuild algorithm, we know that each node in this encrypted index tree is a ciphertext of PO-IPE. Thus, we can use the encrypted vector of $u^{(\beta)}$ in $I_{T^{(\beta)}}$ as the challenge ciphertext of PO-IPE.

(4) Phase 2: $\mathbb{A}$ can continue to ask $\mathbb{C}$ for trapdoors under the restriction described above. Since each trapdoor is generated by using $\text{KeyGen}_{\text{IPE}}$ algorithm, these trapdoors can still be reckoned as a set of decryption keys of PO-IPE.

(5) Response: $\mathbb{A}$ outputs a guess β'. This guess is used as the guess of the security game of PO-IPE.

According to the above game, we can find that the security game in T-PECK is consistent to that in PO-IPE. If this is a PPT adversary $\mathbb{A}$ which can distinguish two encrypted index trees of the T-PECK, we can use $\mathbb{A}$ to distinguish two encrypted vectors of PO-IPE. Thus, we can state that our T-PECK is secure if PO-IPE is secure. □

4.3. Dynamic Update Operations. Previous PECK schemes usually utilized forward index structure, which enables dynamic update for documents inherently. Considering that dynamic update is crucial for the usability of PECK, it is necessary to enable the T-PECK scheme to support some dynamic operations such as document insertion, deletion, and modification. Since utilizing the tree-based index structure, we realize update operations by updating the tree's nodes. Inspired by the update approach given in [16], we devise the update algorithm as follows:

(1) Deletion: if one data owner wants to delete a document f_i from the index tree, he/she first finds the leaf node related to f_i and sets this leaf node as a fake one rather than deleting it. Then, he/she encrypts this fake node and sends it to the cloud server along with the location information of this leaf node. When the cloud server receives this node, it can perform the deletion operation by replacing the target leaf node with the fake one.

(2) Insertion: if a data owner wants to insert a document f_i to the tree, he/she first creates a leaf node for f_i by using Algorithm 4. Then, the data owner will find a fake leaf node and substitute the fake node with the new leaf node. After this, according to the leaf node, the data owner updates all internal nodes on the path from the root node to the new leaf node. Finally, the data owner encrypts the leaf node and the corresponding internal nodes and then sends them along with the position information to the cloud server. When the cloud server receives these nodes, it realizes the insertion operation by replacing these nodes according to the position information. In addition, if there is no fake leaf node that can be replaced, the data owner will generate a new encrypted index tree with many fake leaf nodes and one target leaf node. After receiving this index tree, the cloud server will combine these two index trees as a new one.

(3) Modification: if the data owner wants to modify a leaf node, he/she can delete this leaf node first and then insert a new leaf node according to the modification information.

The keywords in the dictionary will be changed after a period of documents insertion and deletion. Fortunately, this situation does not have much impact on the T-PECK scheme. For the leaf node of a document f_i, the attribute vector of this leaf node is constructed by using the keyword set W_i of f_i, which means that the keyword conversion method can still work well even W_i contains some new keywords. For the "0-1" vector of an internal node, we first review the process of creating a "0-1" vector. The dictionary is divided into a set of keyword sets, and each keyword set is associated with one dimension of the "0-1" vector. When a new keyword is added to the dictionary, we can put this keyword into any keyword set, which will not affect the "0-1" vectors generated before. For example, suppose that the dictionary is $\{w_1, w_2, w_3, w_4, w_5\}$ and is divided into three sets $\{w_1, w_2\}$, $\{w_3, w_4\}$, and $\{w_5\}$ and the new keyword sets are $\{w_1, w_2\}$, $\{w_3, w_4\}$, and $\{w_5, w_6\}$ after adding a keyword w_6 in the dictionary. Note that the "0-1" vectors for $\{w_1, w_5\}$ are both "101" before and after adding w_6. Thus, according to the above analysis, the dynamic update method of our scheme is practicable.

5. Performance Evaluation

We will analyze the theoretical and experimental performance of our scheme in this section by comparing other recent PECK schemes.

5.1. Theoretical Analysis. The theoretical analysis of the proposed T-PECK scheme is evaluated and compared with the previous PECK schemes in terms of space and computation complexity. We choose four representative PECK schemes proposed recently for comparison. For simplicity, we denote these four PECK schemes introduced in [8, 10, 13] by SPE-CKS, SPE-SMKS, SA-SCF-PECKS, and PMSEHS, respectively. For clarity, we define three important parameters associated with these PECK schemes. The first parameter is the number of keywords in an index, denoted by n; the second one is the number of keywords in a query, denoted by m; the third one is the number of documents in a corpus, denoted by d. In addition, for PMSEHS, since the search process only accesses the documents containing the first keyword in the query, we denote the number of documents matching the first keyword in the query by r_1. For T-PECK, we denote the number of keyword sets by k, and the number of documents whose "0-1" vector matches the query vector by r_2 since only these documents will be verified in the search process. According to these denotations, we present Table 3 to show the comparison of time and space complexities among T-PECK and other previous PECK schemes.

As shown in Table 3, the time complexities of search in SPE-CKS, SPE-SMKS, and SA-SCF-PECKS are all linear with d since these three schemes adopt the forward index, while that in our scheme is linear with $\log_2 d$ due to utilizing tree index structure. According to the tree building algorithm (Algorithm 5), we can evaluate that the search algorithm needs at most access $r_2 * \log_2 d$ nodes in the index tree. Since the search algorithm requires the pairing operations three times for each node, T-PECK will perform pairing operations at most $3r_2 * \log_2 d$ times. Because d is a large number and commonly much bigger than n and r_2, we can reckon that the proposed scheme is more efficient than SPE-CKS, SPE-SMKS, and SA-SCF-PECKS in the search process. In addition, because the tree index structure contains many internal nodes to accelerate the search process, the time and storage consumption of our scheme is more than that in SPE-CKS, SPE-SMKS, and SA-SCF-PECKS.

Table 3 also shows that the time complexity of search in PMSEHS is linear with r_1 since it utilizes a hidden structure to build the index. Compared with PMSEHS, the T-PECK scheme needs less search time when $r_2 \log_2 d < r_1$. According to the definition of r_1, r_2, and d, we know our scheme is more efficient when the term frequency of query keywords is high and d is relatively small. Moreover, PMSEHS needs much more time and storage overhead for index generation and storage than T-PECK since the hidden structure requires many group elements of G_T to speed up the query process. As shown in Table 3, the time complexity of index building and the space complexity of index storage in PMSEHS are both linear with $O(n^2)$ while that in T-PECK are both linear with $O(n)$. Thus, we argue that our scheme can achieve a sublinear search complexity without sacrificing the index generation time and storage space.

5.2. Experimental Results. Based on Java Pairing-Based Cryptography (JPBC) library [36], our scheme is implemented on an environment in which the CPU is Intel (R) Core (TM) i7-4570 at 3.60 GHz and the memory size is 16 GB. A type A pair is used to realize the bilinear map in our scheme. The base field size of this pair is 128 bits, and the security level of this pair is equivalent to 1024 bits of DLOG [36]. The real-world corpus that our experiment adopts is an e-mail dataset named by Enron [37]. To quantify the efficiency of our scheme, we mainly focus on three parameters related to PECK in the experiment. These three parameters are n, d, and m mentioned in Section 5.1. To demonstrate the advantages of T-PECK, two previous PECK schemes, that is, SPE-CKS and PMSEHS, are compared with our scheme. The reason for choosing these two schemes is that SPE-CKS adopts a forward index, while PMSEHS utilizes a hidden structure. The comparison experiment involves the time overhead of index building, trapdoor generation, and search.

5.2.1. Impact of the Number of Keywords in a Document (n) on Performance. For a query with 6 keywords ($m = 6$) and a corpus with 300 documents ($d = 300$), as n increases, Figure 4 shows the following facts.

(1) The execution time of index building in PMSEHS far exceeds than that in SPE-CKS and T-PECK since PMSEHS performs lots of pairing operations for accelerating the search process. T-PECK needs more index building time than SPE-CKS since requiring encrypting internal nodes.

(2) The trapdoor generation time in PMSEHS is independent of n and is better than other two scheme for $m < n$. T-PECK needs less time for trapdoor generation than SPE-CKS since it requires less time for exponentiation computation over group elements.

(3) The search complexity in SPE-CKS is linear with n while other two schemes are sublinear with n. As n increases, the search time of PMSEHS and T-PECK grows slightly due to the growth of parameters r_1 and r_2. In addition, the reason why T-PECK needs more time than PMSEHS in search phase is that r_2 grows faster than r_1 as n increases.

5.2.2. Impact of the Number of Documents in a Corpus (d) on Performance. According to the analysis in Section 5.1, the time cost for index building in PMSEHS, SPE-CKS, and T-PECK is linear with d, which is confirmed by the experiment result. PMSEHS costs more time for index building due to requiring more pairing operations for encrypting documents. The time consumption of T-PECK is slightly more than that of SPE-CKS in index building phase since T-PECK needs encrypt extra internal nodes. As shown in

Table 3: Comparison with previous PECK schemes on time and space complexity.

Scheme	$T1$	$T2$	$T3$	$S1$	$S2$
SPE-CKS [8]	$(2n)dP$	$(n)P$	$(n)dE$	$(n)d\|G\|$	$(n)\|G\|$
SPE-SMKS [9]	$(2n+4)dP$	$(n+4)P$	$(n+1)dP+3\ dE$	$(n+3)d\|G\|$	$3\|G_T\|+(n+1)\|Z_P^*\|$
SA-SCF-PECKS [10]	$2\ dE+(n+4)dP$	$(n+2)P$	$4\ dP+(n+2)dE$	$(n+2)d\|G\|+d\|G_T\|$	$(n+2)\|G\|+\|Z_P^*\|$
PMSEHS [13]	$(2n)dP+(n^2)dE$	$(3m)P$	$(2mr_1)E$	$(4\mathrm{nd})\|G\|+n^2d\|G_T\|$	$(2m)\|G\|$
T-PECK	$(2n+2k)dP+2\ dE$	$(n+k)P$	$(\mathrm{nr}_2\log_2 d)P+(3r_2\log_2 d)E$	$(n\ d+nk)\|G\|+2\ d\|G_T\|$	$(3n+3k)\|G\|$
Denotation	G and G_T are two cyclic groups of a large prime order, and $\hat{e}$: $G\times G\longrightarrow G_T$ is a bilinear map [35] P and P_T: the time cost of one exponentiation computation in G, and G_T, respectively E: the time cost of one pairing operation $\|Z_P^*\|$, $\|G\|$ and $\|G_T\|$: the size of an element of $\|Z_P^*\|$, G and G_T, respectively $T1$, $T2$, and T3 represent the time complexity of index building, trapdoor generation, and search, respectively $S1$ and $S2$ represent the space complexity of index and trapdoor, respectively				

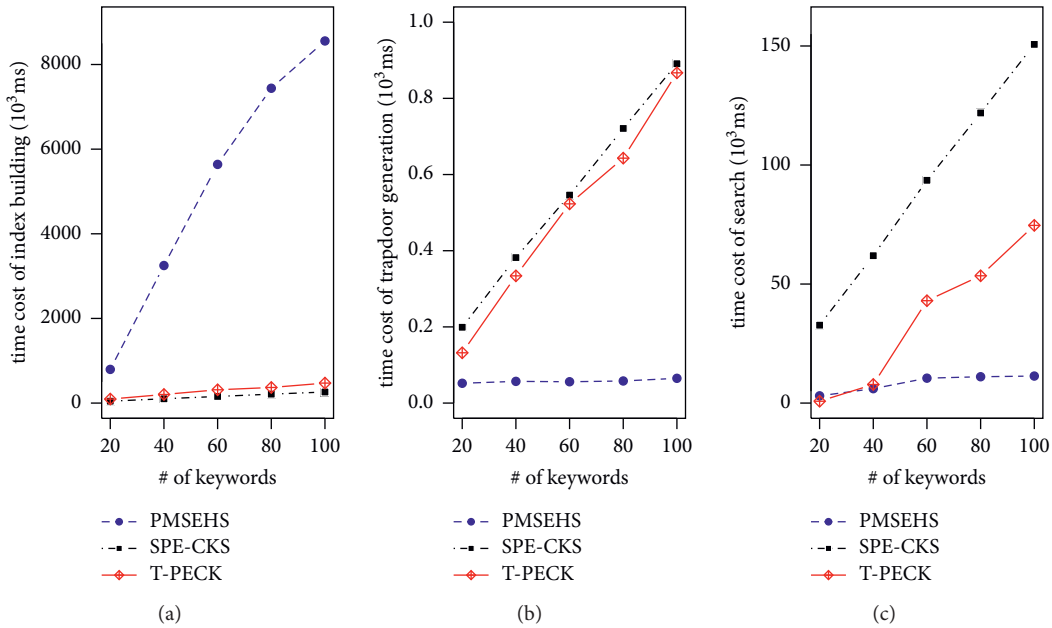

Figure 4: Impact of n on the time cost of index building (a), trapdoor generation (b), and search (c) ($d = 300$, $m = 6$, $k = 60$, and $n = \{20, 40, 60, 80, 100\}$).

Figure 5, we can find that the time cost of trapdoor generation of these three schemes is independent with d. Moreover, Figure 5 also shows that the search complexity in SPE-CKS is linear with d while that in PMSEHS and T-PECK are both sublinear with d. The search time in T-PECK is more than that in PMSEHS, since T-PECK requires more pairing operations than PMSEHS as d increases, which is identical to our theoretical analysis.

5.2.3. Impact of the Number of Keywords in a Query (m) on Performance. According to the analysis in Section 5.1, the parameter m only impacts trapdoor generation and testing. For an index with 60 keywords ($n = 60$), as shown in Figure 6, the time overhead of trapdoor generation in PMSEHS is linearly with $O(m)$, while that in SPE-CKS and T-PECK is independent with m. Moreover, as expected, the time consumption of search in SPE-CKS is independent with m while that in PMSEHS is linear with m. As m increases, T-PECK has better search performance than PMSEHS since the number of documents, whose "0-1" vectors match the query (r_2), is reduced. So, we can say that T-PECK is more efficient than PMSEHS when m is large.

5.3. More Discussion. According to experimental results, when $n = 60$, $d = 500$, and $m = 6$, the time consumption of search in T-PECK is 41 s while that in SPE-CKS is 162 s. As compensation, the time overhead of index building in T-PECK is 460 s while that in SPE-CKS is 265 s. According to this result, we can argue that the search performance of our scheme is better than that of SPE-CKS without sacrificing the time complexity of index building. Compared with PMSEHS, when $n = 60$, $d = 500$, and $m = 6$, the time cost of index building in T-PECK is 460 s while that in PMSEHS is 9538 s. Accordingly, our scheme costs twice as much search time as PMSEHS. Thus, our scheme requires much less time cost in the index building process while ensuring the sublinear search efficiency. In practice, the index building

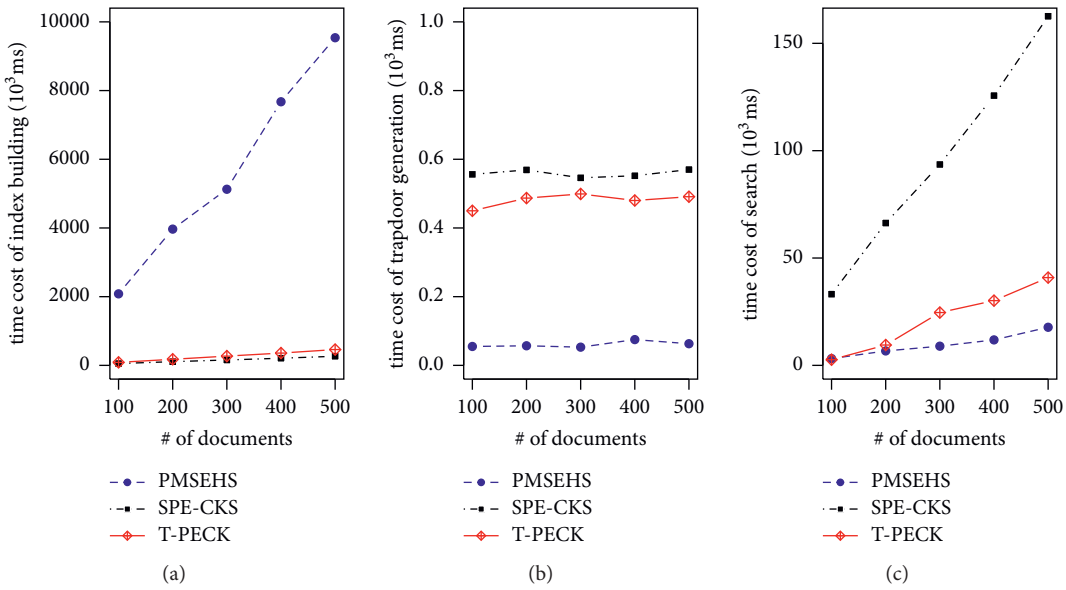

Figure 5: Impact of d on the time cost of index building (a), trapdoor generation (b), and search (c) ($n = 60$, $m = 6$, $k = 60$, and $d = \{100, 200, 300, 400, 500\}$).

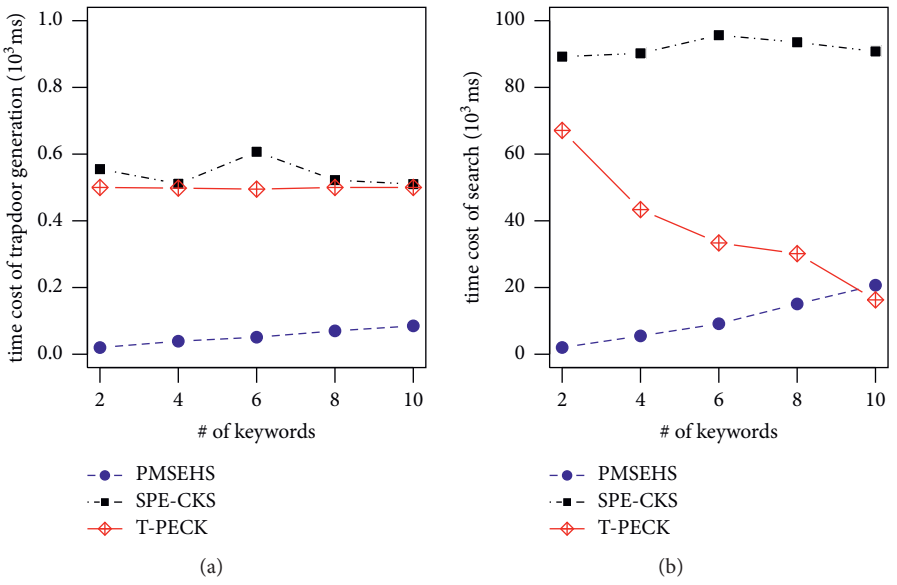

Figure 6: Impact of m on the time cost of trapdoor generation (a) and search (b) ($D = 300$, $n = 60$, $k = 60$, and $m = \{2, 4, 6, 8, 10\}$).

algorithm is usually performed by data owners while the search algorithm is run by the cloud server. Thus, considering that the cloud server owns much more computing and storage resources than data owner, we reckon that it is worth to sacrifice a little query efficiency to reduce time and space costs of index building and storage.

To summarize, it is clear that our scheme maintains a high query efficiency without increasing the time cost of index generation too much. Considering the fact that our scheme holds a good trade-off between query and index generation, we reckon that T-PECK is practicable in applications in which data users use resource-constrained mobile devices.

6. Conclusion

In this paper, we proposed a novel algorithm for building an index tree. Through elaborately combining the index tree and an efficient PO-IPE scheme, we proposed a PECK scheme based on a tree-based index structure. The search efficiency in the proposed scheme is sublinear with the number of documents, and our scheme is proven to be L-semantically secure against chosen plaintext attacks.

To evaluate the efficiency of T-PECK, a detailed theoretical and experimental analysis is proposed. This analysis shows that compared with previous PECK schemes, T-PECK is more practical such as requiring less time for

index building and search. In real-world applications, the query of data users is usually more complex than conjunctive keyword search, such as Boolean keyword search, fuzzy search, and range search. Thus, it is necessary to build a tree-based SPE scheme supporting more expressive search function.

Data Availability

The data used to support the findings of this study are available from "http://www.cs.cmu.edu/./enron/."

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant nos. 61972090 and 31872704), Natural Science Foundation of Henan (Grant no. 202300410339), and Nanhu Scholars Program for Young Scholars of XYNU.

References

[1] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searching on encrypted data," in *Proceedings of the IEEE Symposium on Research in Security and Privacy 2000. Los Alamitos*, pp. 44–55, IEEE Computer Society Press, Berkeley, CA, USA, May 2000.

[2] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, C. Cachin and J. L. Camenisch, Eds., pp. 506–522, Springer, Interlaken, Switzerland, May 2004.

[3] D. J. Park, K. Kim, and P. J. Lee, "Public key encryption with conjunctive field keyword search," Edited by C. H. Lim and M. Yung, Eds., in *Proceedings of the International Workshop on Information Security Applications*, vol. 3325, Springer, Jeju Island, Republic of Korea, August 2004.

[4] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of the TCC 2007*, S. P. Vadhan, Ed., vol. 4392, pp. 535–554, Springer, Amsterdam, The Netherlands, February 2007.

[5] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 262–267, 2011.

[6] C. Song, X. Liu, and Y. Yan, "Efficient public key encryption with field-free conjunctive keywords search," in *Proceedings of the Revised Selected Papers of the 6th International Conference on Trusted SystemsTrusted Systems*, pp. 394–406, Springer International Publishing, Beijing China, December 2014.

[7] Y. Yang, X. Liu, and R. Deng, "Expressive query over outsourced encrypted data," *Information Sciences*, vol. 442-443, pp. 33–53, 2018.

[8] Y. Zhang, Y. Li, and Y. Wang, "Efficient conjunctive keywords search over encrypted E-mail data in public key setting," *Applied Sciences*, vol. 9, no. 18, p. 3655, 2019.

[9] Y. Zhang, Y. Wang, and Y. Li, "Searchable public key encryption supporting semantic multi-keywords search," *IEEE Access*, vol. 7, Article ID 122078, 2019.

[10] R. Zhang, J. Wang, Z. Song, and X. Wang, "An enhanced searchable encryption scheme for secure data outsourcing," *Science China Information Sciences*, vol. 63, no. 3, pp. 1–16, 2020.

[11] Y. Miao, X. Liu, K. K. R. Choo et al., "Privacy-preserving attribute-based keyword search in shared multi-owner setting," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, 2019.

[12] Y. Lu and J. Li, "Constructing pairing-free certificateless public key encryption with keyword search," *Frontiers of Information Technology & Electronic Engineering*, vol. 20, no. 8, pp. 1049–1060, 2019.

[13] P. Xu, S. Tang, P. Xu, Q. Wu, H. Hu, and W. Susilo, "Practical multi-keyword and Boolean search over encrypted e-mail in cloud server," *IEEE Transactions on Services Computing*, 2019.

[14] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 340–352, 2016.

[15] X. Ding, P. Liu, and H. Jin, "Privacy-Preserving multi-keyword top-k similarity search over encrypted data," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 344–357, 2017.

[16] H. Zhong, Z. Li, J. Cui, Y. Sun, and L. Liu, "Efficient dynamic multi-keyword fuzzy search over encrypted cloud data," *Journal of Network and Computer Applications*, vol. 149, Article ID 102469, 2020.

[17] F. Boucenna, O. Nouali, and S. Kechid, "Secure inverted index based search over encrypted cloud data with user access rights management," *Journal of Computer Science and Technology*, vol. 34, no. 1, pp. 133–154, 2019.

[18] M. Yang, H. Dai, J. Bao, X. Yi, and G. Yang, "A parallel multi-keyword top-k search scheme over encrypted cloud data," in *Proceedings of the IFIP International Conference on Network and Parallel Computing*, pp. 169–181, Springer, Hohhot, China, August 2019.

[19] I. Kim, S. O. Hwang, J. H. Park, and C. Park, "An efficient predicate encryption with constant pairing computations and minimum costs," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 2947–2958, 2016.

[20] C. Gentry, "Practical identity-based encryption without random oracles," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 445–464, Springer, St. Petersburg, Russia, May 2006.

[21] M. Abdalla, M. Bellare, D. Catalano et al., "Searchable encryption revisited: consistency properties, relation to anonymous IBE, and extension," in *Proceedings of the CRYPTO 2005, Lecture Notes in Computer Science*, vol. 3621, pp. 205–222, Springer, Berlin, Heidelberg, 2005.

[22] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," *Journal of Cryptology*, vol. 26, no. 2, pp. 191–224, 2013.

[23] T. Okamoto and K. Takashima, "Achieving short ciphertexts or short secret-keys for adaptively secure general inner-product encryption[J]," *Designs, Codes and Cryptography*, vol. 77, pp. 138–159, 2015.

[24] Y. Zhang and S. Lu, "POSTER: efficient method for disjunctive and conjunctive keyword search over encrypted data [J]," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1535–1537, Association for Computing Machinery, Scottsdale Arizona USA, November 2014.

[25] Y. Zhang, Y. Li, and Y. Wang, "Secure and efficient searchable public key encryption for resource constrained environment based on pairings under prime order group," *Security and Communication Networks*, vol. 2019, Article ID 5280806, 14 pages, 2019.

[26] Y. Zhang, Y. Zhao, Y. Wang, and Y. Li, "Searchable public key encryption supporting simple boolean keywords search," *IEICE - Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E103.A, no. 1, pp. 114–124, 2020.

[27] J. W. Byun, H. S. Rhee, H.-A. Park, and D. H. Lee, "Off-line keyword guessing attacks on recent keyword search schemes over encrypted data," in *Workshop on Secure Data Management*, W. Jonker and M. Petković, Eds., Springer, Berlin, Heidelberg, pp. 75–83, 2006.

[28] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee, "Trapdoor security in a searchable public-key encryption scheme with a designated tester," *Journal of Systems and Software*, vol. 83, no. 5, pp. 763–771, 2010.

[29] Q. Tang and L. Chen, "Public-key encryption with registered keyword search," in *Proceedings of the European Public Key Infrastructure Workshop*, F. Martinelli and B. Preneel, Eds., Springer, Pisa, Italy, pp. 163–178, September 2009.

[30] H. Zhu, Z. Mei, B. Wu, H. Li, and Z. Cui, "Fuzzy keyword search and access control over ciphertexts in cloud computing," in *Proceedings of the Australian Conference on Information Security and Privacy*, Springer, Auckland, New Zealand, July 2017.

[31] J. Li, X. Lin, Y. Zhang, and J. Han, "KSF-OABE: outsourced attribute-based encryption with keyword search function for cloud storage," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 715–725, 2017.

[32] P. Xu, Q. Wu, W. Wang, W. Susilo, J. F. Domingo, and H. Jin, "Generating searchable public-key ciphertexts with hidden structures for fast keyword search[J]," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 9, pp. 1993–2006, 2017.

[33] P. Xu, S. He, W. Wang, W. Susilo, and H. Jin, "Lightweight searchable public-key encryption for cloud-assisted wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3712–3723, 2017.

[34] Y. Zhang, Y. Li, and Y. Wang, "Conjunctive and disjunctive keyword search over encrypted mobile cloud data in public key system," *Mobile Information Systems*, vol. 2018, Article ID 3839254, 11 pages, 2018.

[35] A. Joux, "The weil and tate pairings as building blocks for public key cryptosystems," in *Proceedings of the International Algorithmic Number Theory Symposium*, C. Fieker and D. R. Kohel, Eds., vol. 2369, pp. 20–32, Springer, Sydney, Australia, July 2002.

[36] A. D. Caro, "The java pairing based cryptography library (jpbc)," 2013, http://gas.dia.unisa.it/projects/jpbc/download.html#.YXqT5FVBzIU.

[37] W. W. Cohen, "Enron E-mail dataset," http://www.cs.cmu.edu/%7E./enron/.