IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received November 1, 2019, accepted November 13, 2019, date of current version November 26, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2954043

Achieving Efficient and Privacy-Preserving Multi-Keyword Conjunctive Query Over Cloud

FAN YIN[1,2], YANDONG ZHENG[2], RONGXING LU[2], (Senior Member, IEEE), AND XIAOHU TANG[1], (Senior Member, IEEE)

[1]Information Security and National Computing Grid Laboratory, Southwest Jiaotong University, Chengdu 611756, China
[2]Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada

Corresponding authors: Rongxing Lu (rlu1@unb.ca) and Xiaohu Tang (xhutang@swjtu.edu.cn)

This work was supported in part by the NSERC Discovery under Grant Rgpin 04009, in part by the NBIF Start-Up under Grant Rif 2017-012, in part by the HMF under Grant 2017 YS-04, in part by the URF under Grant Nf-2017-05, in part by the LMCRF-S-2018-03, and in part by the NSFC under Grant 61871331.

ABSTRACT With the explosive growth of data, it has become increasingly popular to deploy the powerful cloud to manage data. Meanwhile, as the cloud is not always fully trusted, personal and sensitive data have to be encrypted before being outsourced to the cloud. Naturally, this brings a serious challenge for the cloud to provide secure and efficient query services over huge volumes of data. Although existing works have proposed some solutions to solve the above challenge, most of them just focus on the single keyword query and cannot directly support multi-keyword query. Even though some works have discussed solutions for the multi-keyword query, they cannot well balance the efficiency and privacy. Therefore, in this paper, we propose a novel multi-keyword conjunctive query scheme over cloud, which can achieve high query efficiency with small privacy leakage. In specific, we first design a tree-based index to support the multi-keyword conjunctive query and employ Boneh-Goh-Nissim (BGN) homomorphic encryption technique to protect its privacy. Then, based on the tree-based index, we propose a wildcard search algorithm to improve its query efficiency. Finally, the detailed security analysis shows that the proposed scheme is really privacy-preserving, and extensive simulation results also demonstrate its efficient.

INDEX TERMS Cloud computing, conjunctive query, homomorphic encryption, multi-keyword query.

I. INTRODUCTION

With the rapid development of the internet, volumes of data are exploding by the day. According to IBM Marketing Cloud study [1], more than 90% of data on the internet has been created since 2016, which leads more and more individuals and companies to store local files in the cloud to reduce the increasing storage overhead. However, the cloud servers may not be fully trusted in practice today, because administrators or even hackers are likely to get full access to the servers and consequently to the files. Thus, the files with some sensitive information (e.g., electronic health records) have to be encrypted before outsourcing them to the cloud. Although data encryption technique preserves data privacy, it also hides some critical information such that the cloud cannot support some user's operations over the encrypted data, e.g., as multi-keyword conjunctive query, which returns a set of files containing multiple queried keywords. Consequently, it is challenging to perform multi-keyword conjunctive query over encrypted data.

One straightforward solution is that the user downloads all encrypted files and performs search after files are decrypted. However, this solution is impractical because of its significant computational cost and communication overhead. In order to solve the above problem, searchable encryption (SE) was introduced, which allows the cloud server to search encrypted files without leaking information in the plaintext files. SE can be realized with optimal security via powerful cryptographic tools, such as Fully Homomorphic Encryption (FHE) [2] and Oblivious Random Access Memory (ORAM) [3]. However, these tools are also impractical and sometimes may even be slower than the naive solution. Another set of works [4]

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

utilize Property-preserving Encryption (PPE) to construct their schemes. Nevertheless, these schemes based on PPE inevitably leak certain properties of the underlying message (e.g., order, frequency), which makes them vulnerable to statistical attacks.

For balancing the leakage and efficiency, many studies [5]–[7] focus on Symmetric Searchable Encryption (SSE), which relaxes the security of FHE and ORAM by leaking the access pattern (i.e., search result) and search pattern (i.e., queries have the same queried keywords). But nothing else is leaked (e.g., order). Unfortunately, most studies only consider single keyword query, which can not be used directly to achieve multi-keyword conjunctive queries. A straightforward method is to query each keyword first. Then, the cloud server finds a set of files that only matches each keyword and returns the intersection of all these sets. However, this method is flawed because it allows the cloud server to learn much extra information in addition to the results of the multi-keyword conjunctive query. For example, the cloud server can observe which files contain a certain keyword. Recently, some privacy-preserving multi-keyword conjunctive query schemes [8]–[10] are proposed to support multi-keyword conjunctive search service with small leakage. However, these schemes are still not perfect enough, since they can not achieve efficient query.

To address the above problems, in this paper, we propose a new privacy-preserving multi-keyword conjunctive query scheme, which can well balance the efficiency and privacy in query. Specifically, the main contributions of this work are three-fold:

- First, we design a tree-based index, called conjunctive tree, to represent keywords. With this conjunctive tree, the server can efficiently conduct multi-keyword conjunctive queries. At the same time, we employ BGN homomorphic encryption technique to encrypt the conjunctive tree, which can well preserve the privacy of keywords.
- Second, we design a novel search algorithm, called wildcard search algorithm, for the conjunctive tree to improve the efficiency of conjunctive query, which replaces a part of computationally expensive conjunctive tree traversing operations with efficient string copy operations.
- Third, we analyze the security of our proposed scheme and conduct extensive experiments to evaluate its performance, which indicate that our proposed scheme can achieve efficient query with small leakage.

The remainder of this paper is organized as follows. In Section II, we introduce our system model, security model and design goal. Then, we describe some preliminaries in Section III. In Section IV, we present our proposed scheme, followed by security analysis and performance evaluation in Section V and Section VI, respectively. Related work is discussed in Section VII. Finally, we draw our conclusions in Section VIII.

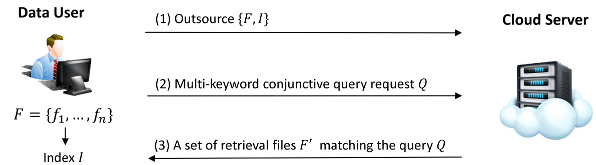


FIGURE 1. System model under consideration.

II. MODELS AND DESIGN GOALS

In this section, we formalize the system model, security model, design goals, and notations.

A. SYSTEM MODEL

In our system model, we consider a typical single user/single server searchable encryption model which consists of two entities, as shown in Fig. 1.

- **Data user**: We assume the data user has a collection of files $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ and each file $f_j \in \mathcal{F}$ consists of a set of keywords from a keyword dictionary $\mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_d\}$. Due to the limited storage space and computational capability, the data user intends to outsource the file collection $\mathcal{F}$ and its index I to the cloud server. Then, the data user submits a multi-keyword conjunctive query request Q generated from a set of queried keywords and retrieves files containing all these queried keywords from the cloud server. Note that, since the $\mathcal{F}$, I and Q are private information, they need to be encrypted before being sent to the cloud server.
- **Cloud server**: The cloud server is considered to be powerful in storage space and computational capability. The duties of the cloud server include: i) efficiently store file collection $\mathcal{F}$ and index I, and ii) process multi-keyword conjunctive query request Q and respond all the matching files $\mathcal{F}'$ to the data user.

B. SECURITY MODEL

In our security model, we consider attacks from two types of entities, one of which is the cloud server. Concretely, we consider the cloud server to be honest-but-curious, which means that it honestly executes the query processing and returns the query results without tampering it. However, the cloud server tries to infer as much information as possible from the available data which include encrypted files, indexes, and the encrypted query requests. In addition to the cloud server, we also consider attacks from an outside entity who can eavesdrop on the data transmitted during the query. The outside entity is curious about the information of the queries. For example, he/she may compare two query requests to determine whether they have the same queried keywords.

Since our work is focused on the efficiency of communication and computation in privacy-preserving query, other active attacks on data integrity and source authentication are beyond the scope of this paper and will be discussed in our future work, although it is not difficult to apply some mature digital signature and message authentication code techniques to tackle these attacks.

TABLE 1. Summary of Notations.

Symbol	Description
$\mathcal{W} = \{\omega_1, \omega_2, ..., \omega_d\}$	The keyword dictionary
$\mathcal{F} = \{f_1, f_2, ..., f_n\}$	The file collection
$\mathcal{W}_j \subseteq \mathcal{W}$	The set of keywords included in $f_j \in \mathcal{W}$
$\mathcal{V}_j$	The file vector of $f_j \in \mathcal{W}$
$N = \{N_0, N_1, ..., N_{2^d-1}\}$	The set of leave in the conjunctive tree
P_l	The path vector of $N_l \in N$
P'_l	The permuted path vector of $N_l \in N$

C. DESIGN GOALS

In this paper, our design goal is to achieve an efficient and privacy-preserving multi-keyword conjunctive query scheme. The details are described as follows:

- *Privacy-preservation*. In the proposed scheme, the data stored in the cloud server, $\mathcal{F}$ and I, should be privacy-preserving during the query and update.
- *Efficiency*. In order to achieve the above privacy requirement, additional computational cost will be incurred. Specifically, we aim to reduce this cost as much as possible and make the proposed scheme efficient than previous work.

D. NOTATIONS

See Table 1.

III. PRELIMINARY

In this section, we outline the bilinear groups of composite order and the BGN homomorphic encryption [11], which will serve as the basis of our proposed scheme.

A. BILINEAR GROUPS OF COMPOSITE ORDER

Given a security parameter κ, a composite bilinear parameter generator $Gen(\kappa)$ outputs a tuple (N, g, G, G_T, e), where $N = pq$ and p, q are two κ-bit prime number, G and G_T are two finite cyclic multiplicative groups of composite order N, $g \in G$ is a generator, and $e : G \times G \rightarrow G_T$ is a bilinear map with the following properties:

- *Bilinearity* $e(g^a, h^b) = e(g, h)^{ab}$ for any $(g, h) \in G^2$ and $a, b \in Z_N$.
- *Nondegeneracy* If g is a generator of G, then $e(g, g)$ is a generator of G_T with order N.
- *Computability* There exists an efficient algorithm to compute $e(g, h) \in G_T$ for all $(g, h) \in G$.

Let g be a generator of G, then $g' = g^q \in G$ can generate the subgroup $G_p = \{g'^0, g'^1, \ldots, g'^{p-1}\}$ of order p, and $g'' = g^p \in G$ can generate the subgroup $G_q = \{g''^0, g''^1, \ldots, g''^{q-1}\}$ of order q in G. The SubGroup Decision (SGD) Problem in G is stated as follows [11]: given a tuple (N, g, G, G_T, e, h), where the element h is drawn randomly from either G or subgroup G_q, decide whether $h \in G_q$ or not. When we assume that the SGD problem is hard, the security of the BGN homomorphic encryption can be ensured [11].

B. BGN HOMOMORPHIC ENCRYPTION

The Boneh-Goh-Nissim (BGN) homomorphic encryption includes three algorithms: key generation, encryption, and decryption. The details are described as follows:

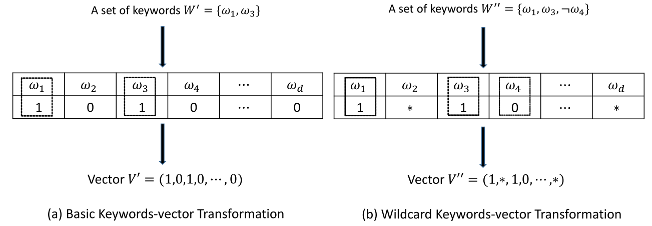


FIGURE 2. Examples of keywords-vector transformations.

- *Key Generation:* Given a security parameter κ, run $Gen(\kappa)$ to get a tuple (N, g, G, G_T, e) as described in Section III-A. The $g \in G$ is a generator of order N, $N = pq$ and p, q are κ-bit prime numbers. Set $h = g^q$, then h is a random generator of the subgroup of G of order p. The private key is $sk = p$ and the corresponding public key is $pk = (N, G, G_T, e, g, h)$.
- *Encryption:* We assume the message space consists of integers in the set $S = \{0, 1, \ldots, \Delta\}$, the size of set S is application-oriented and much smaller than q, i.e., $\Delta \ll q$. To encrypt a message $m \in S$, we choose a random number $r \in Z_N$, and compute the ciphertext $c = E(m, r) = g^m h^r \in G$.
- *Decryption:* Given the ciphertext $c = E(m, r) = g^m h^r \in G$, the corresponding message can be recovered by the private key p. Observe that $c^p = (g^m h^r)^p = (g^p)^m$, we can set $\hat{g} = g^p$. Then, to recover m, it suffices to compute the discrete log of c^p base $\hat{g}$. Since $0 \leq m \leq \Delta$, the expected time is around $O(\sqrt{\Delta})$ when using the Pollard's lambda method [12] (p.128).

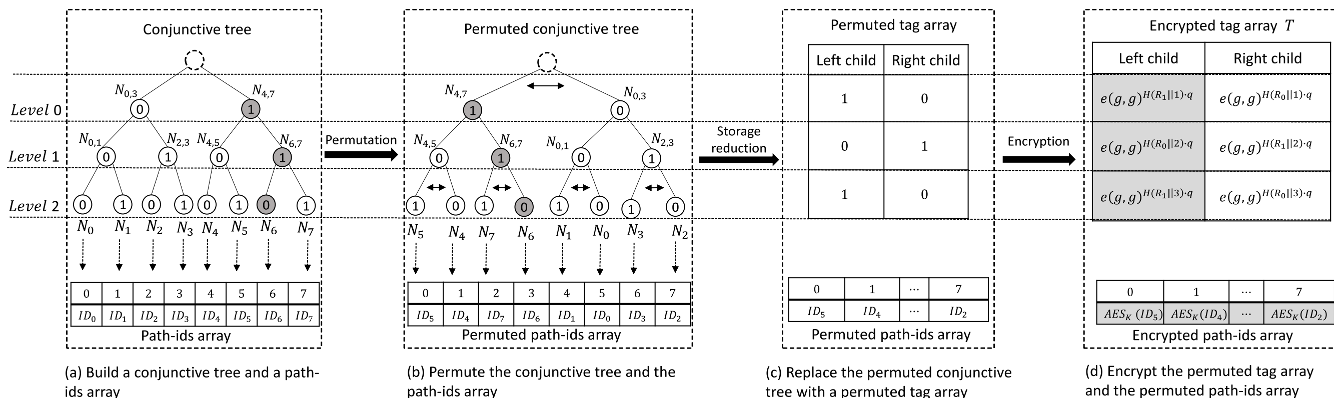
IV. THE PROPOSED SCHEME

In this section, we will present our new privacy-preserving multi-keyword conjunctive query scheme. Before delving into the details, we first introduce two keywords-vector transformation methods which are basic building blocks of the proposed scheme.

A. KEYWORDS-VECTOR TRANSFORMATION METHODS

In order to accurately describe the construction of index and query requests, we define two keywords-vector transformation methods, which transform a set of keywords to a $1 \times d$ vector. Let $\mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_d\}$ be a keyword dictionary, and then the two methods are described as follows:

- *Basic keywords-vector transformation method:* The first method transforms a set of keywords $\mathcal{W}' \subseteq \mathcal{W}$ to a $1 \times d$ vector $V' = \{k_0, k_1, \ldots, k_{d-1}\}$, where $k_i \in \{0, 1\}$. This method will be used to transfer a file to a file vector in our proposed scheme later. Specifically, it first initializes a $1 \times d$ vector $V' = (0, 0, \ldots, 0)$. Then, it checks the set of keywords $\mathcal{W}'$ and sets $k_{i-1} = 1$ for each $\omega_i \in \mathcal{W}'$. As shown in Fig. 2(a), a set of keywords $\mathcal{W}' = \{\omega_1, \omega_3\}$ is transformed to a vector $V' = (1, 0, 1, 0, \ldots, 0)$, where $k_0, k_2 = 1$ and other elements are set to 0.
- *Expressive keywords-vector transformation method:* The second method transforms a set of keywords



(a) Build a conjunctive tree and a path-ids array

(b) Permute the conjunctive tree and the path-ids array

(c) Replace the permuted conjunctive tree with a permuted tag array

(d) Encrypt the permuted tag array and the permuted path-ids array

FIGURE 3. An example of index construction with height $d = 3$. (a) Build a conjunctive tree (perfect binary tree) with $2^d = 2^3 = 8$ leaf nodes. Each root-to-leaf path in the conjunctive tree is associated with a set of file identifiers whose keywords satisfy the path. For example, all the files in ID_6 have the keyword set $W_6 = \{\omega_1, \omega_2\}$. (b) Randomly permute the sibling nodes in conjunctive tree where the permutations at the same level follow the same rule. (c) Replace the permuted conjunctive tree with a permuted tag array to save the storage. (d) Encrypt the tags in the permuted tag array T through the secret keys generated in the system initialization.

$\mathcal{W}'' \subseteq \mathcal{W}$ to a $1 \times d$ vector $V'' = \{k_0, k_1, \ldots, k_{d-1}\}$, where $k_i \in \{0, 1, *\}$. This method will be used to transfer a multi-keyword query to a query vector in our proposed scheme later. Different from the basic one, this method considers more complex situation that the set of keywords $\mathcal{W}'' = \{\omega_1, \omega_3, \neg\omega_4\}$ concerns not only the keywords (e.g., ω_1, ω_3) but also the *NOT* queried keywords (e.g., $\neg\omega_4$). For each $\omega_i \in \mathcal{W}''$, k_{i-1} is set to 1, and for each $\neg\omega_j \in \mathcal{W}''$, k_{j-1} is set to 0. At the same time, the rest of keywords in the keyword dictionary, i.e., $\mathcal{W} \setminus \mathcal{W}''$, are called wildcard keywords. For each wildcard keyword $\omega_z \in \mathcal{W} \setminus \mathcal{W}''$, k_{z-1} is set to $*$. As seen in Fig. 2, $\mathcal{W}'' = \{\omega_1, \omega_3, \neg\omega_4\}$ can be transformed to $V'' = \{1, *, 1, 0, \cdots, *\}$ where the k_0, k_2 are set to be 1, k_3 are set to 0, and other elements are set to $*$.

B. DESCRIPTION OF OUR PROPOSED SCHEME

In this subsection, we describe our proposed query scheme, which mainly consists of four phases: i) System Initialization; ii) Local File Outsourcing; iii) Multi-keyword Conjunctive Query; and iv) File Update.

1) SYSTEM INITIALIZATION

Given a security parameter κ, the data user first runs $Gen(\kappa)$ algorithm of BGN cryptosystem to get the public key $pk = (N, G, G_T, e, g, h)$ and the corresponding private key $sk = p$. Next, the data user chooses the Advanced Encryption Standard (AES) algorithm as the basic encryption algorithm and selects a random number K as its secret key. Further, the data user initializes a one-way hash function $H : \{0, 1\}^* \longrightarrow Z_N$ and chooses two random numbers R_0, R_1 from Z_N. Finally, the data user keeps $\{K, sk, R_0, R_1\}$ and keyword dictionary $\mathcal{W}$ secret (stored in local), then sends $\{pk, H\}$ to the cloud server.

2) LOCAL DATA OUTSOURCING

Consider the data user has a file collection $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, where each $f_j \in \mathcal{F}$ includes a set of keywords $\mathcal{W}_j \subseteq \mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_d\}$ and an unique identifier id_j. Then the data user builds an encrypted index and an encrypted file collection as following steps:

Step 1: For each file $f_j \in \mathcal{F}$, the data user transforms its $\mathcal{W}_j$ to a $1 \times d$ vector $V_j = (k_0, k_1, \ldots, k_{d-1})$, called its file vector, through the basic keywords-vector transformation method mentioned before (see IV-A).

Step 2: In order to support efficient conjunctive query, the data user builds an index, which consists of a tree and an array, to support efficient conjunctive query.

- First, the data user builds a full binary tree with d height, called conjunctive tree (see Fig. 3(a)), to represent all the files in $\mathcal{F}$. In specific, the conjunctive tree stores a bit $t \in \{0, 1\}$, called tag, for each node (except the root): $t = 0/1$ if the node is left/right child of its parent. Based on these tags, the conjunctive tree labels each leaf N_l $(0 \leq l \leq 2^d - 1)$ with a $1 \times d$ binary vector $P_l = (t_0, t_1, \ldots, t_{d-1})$, called a path vector, which consists of d tags stored in the nodes along the N_l's root-to-leaf path (except the root). Then, each path vector $P_l = (t_0, t_1, \ldots, t_{d-1})$ is seen as a file vector, in which $t_i = 1/0$ means the keyword ω_i is included/not included in the file. In this way, the conjunctive tree associates each leaf N_l with a set of files, whose identifiers are expressed as $ID_l = \{id_j | f_j \in \mathcal{F}, V_j = P_l\}$. For the example in Fig. 3(a), given a dictionary $\mathcal{W} = \{\omega_1, \omega_2, \omega_3\}$, the data user builds a conjunctive tree with height $d = 3$, which associates the leaf N_6 (shown in gray) with a set of files, whose identifiers form the ID_6.
- Second, in order to support the wildcard search algorithm described later (see Alg. 1), the data user builds a 1×2^d array, called path-ids, to store all the sets of file identifiers. Specifically, the path-ids stores ID_l $(0 \leq l \leq 2^d - 1)$ in the l location.

Step 3: In order to protect the privacy of path vectors, the data user transforms the conjunctive tree and the path-ids array to a permuted conjunctive tree and a permuted path-ids array as follows:

Algorithm 1 Search(Encrypted Query Vector $E(Q)$, Encrypted Tag Array T)

```
1:  initialize a wildcard offset stack s
2:  initialize a 1 × d path vector P
3:  initialize a path vector set P̄
4:  for each i in [0, ..., d − 1] do                ▷ Basic search
5:      if E(k_i) == * then
6:          s.push(i); P[i] = 0
7:      else if T[i] == e(E(k_i), h) then
8:          P[i] = 0
9:      else
10:         P[i] = 1
11: P̄ = {P}
12: while stack s is not empty do                   ▷ wildcard search
13:     i = s.top(); s.pop();
14:     copy all elements in P̄ to a new vector set P̄'
15:     for each P' in P̄' do
16:         P'[i] = 1
17:     P̄ = P̄ ∪ P̄'
18: return P̄
```

- First, the data user randomly permutes the locations of sibling nodes in the conjunctive tree. Specifically, the data user chooses a number $\delta_i \in \{0, 1\}$ for each level i $(0 \leq i \leq d-1)$. If $\delta_i = 1$, the data user permutes the sibling nodes at the level i; otherwise, no permutation performs. Note that, the permutations at the same level must follow the same rule, which is necessary for the cloud server to perform wildcard search (see Alg. 1). After these random permutations, the path vector of each leaf N_l is transformed from $P_l = (t_0, t_1, \ldots, t_{d-1})$ to a permuted path vector $P'_l = (t'_0, t'_1, \ldots, t'_{d-1})$, where $t'_i = t_i \oplus \delta_i$ $(0 \leq i \leq d-1)$. As seen in Fig. 3(a-b), the data user randomly chooses $(\delta_1, \delta_2, \delta_3) = (1, 0, 1)$ and then conducts permutations at the level 0 and level 2. After these permutations, the path vector $P_6 = (1, 1, 0)$ of N_6 is transformed to $P'_6 = (0, 1, 1)$ (shown in gray).
- Second, the data user permutes the locations of elements in the path-ids array according to the permutations in conjunctive tree; that is, the location of ID_l $(0 \leq l \leq 2^d - 1)$ in permuted array follows the location of N_l in permuted conjunctive tree. As shown in Fig. 3(a-b), after permutations, the ID_6 in path-ids array is moved from the 6-th location to the 3-th location, which is corresponding to N_6.

Step 4: Since the permuted conjunctive tree permutes sibling nodes at the same level with the same rule, each level on the tree only has two tag values, one for left children and the other for right children. Therefore, the data user can reduce the permuted conjunctive tree to a $d \times 2$ array, called a permuted tag array. In specific, each element stored in the permuted tag array can be expressed as $t_{i,s} \in \{0, 1\}$, where $i \in [0, d-1]$ represents the level number and $s \in \{l, r\}$ represents either left $(s = l)$ or right $(s = r)$ children. For example, in Fig. 3(b-c), since the permuted conjunctive tree stores tag 1 in the left children at level 0, permuted tag array sets $t_{0,l} = 1$.

Step 5: In order to protect the privacy of path vectors and file identifiers, the data user encrypts the permuted tag array and permuted path-ids array as follows:

- For each $t_{i,s} \in \{0, 1\}$ in the permuted tag array, the data user encrypts it as
$$c_{i,s} = e(g, g)^{H(R_{t_{i,s}}||i)\cdot q} \in G_T \tag{1}$$
where R_0 and R_1 are two random numbers only known by the data user. Note that, the data user just outsources one column of the encrypted tag array, e.g., left children (shown in gray), to the cloud server, which is enough for the cloud server to conduct query processing.
- For each element ID_l $(0 \leq l \leq 2^d - 1)$ in the permuted path-ids array, the data user uses AES encryption to encrypt it as $AES_K(ID_l)$.

Step 6: Finally, the data user encrypts each file $f_j \in \mathcal{F}$ through AES encryption and sends these encrypted files to the cloud server with the encrypted tag array and encrypted path-ids array.

3) CONJUNCTIVE QUERY

Given a keyword set $\mathcal{W}' \subseteq \mathcal{W}$, which includes queried keywords and *NOT* queried keywords, the data user launches a privacy-preserving multi-keyword conjunctive query with the cloud server in the following steps:

Step 1: The data user first transforms $\mathcal{W}'$ to a query vector $Q = (k_0, k_1, \ldots, k_{d-1}) \subseteq \{0, 1, *\}^d$ through the expressive keywords-vector transformation method. Then the data user chooses d random numbers $r_0, r_1, \ldots, r_{d-1}$ from the Z_N, encrypts the Q as $E(Q) = (E(k_0), E(k_1), \ldots, E(k_{d-1}))$, where

$$E(k_i) = \begin{cases} g^{H(R_{k_i}||i)+p\cdot r_i}, & \text{if } k_i \neq * \\ *, & \text{if } k_i = * \end{cases} \tag{2}$$

and sends $E(Q)$ to the cloud server.

Step 2: After receiving the $E(Q)$, the cloud server authenticates the query request and rejects the illegal one.

Step 3: If the query request passes the authentication, the cloud server performs the search algorithm 1 to find the matching leaves. The details are as follows:

- *Initialization:* This algorithm initializes an empty stack s to store the locations of wildcard keywords in the query, a $1 \times d$ vector P to store the first matching result, and a set $\overline{\mathbb{P}}$ to store all the matching results (lines 1-3).
- *Basic search:* For each i $(0 \leq i \leq d-1)$, the algorithm computes $e(E(k_i), h)$ and compares it with the i-th element of the encrypted tag array T. If $e(E(k_i), h) = T[i]$, it sets $P[i] = 0$; otherwise, it sets $P[i] = 1$. Note that, if there is a wildcard keyword in the query, the algorithm records its location to the stack s and sets $P[i] = 0$ without comparison. Finally, the algorithm inserts P to the $\overline{\mathbb{P}}$ (lines 4-11).

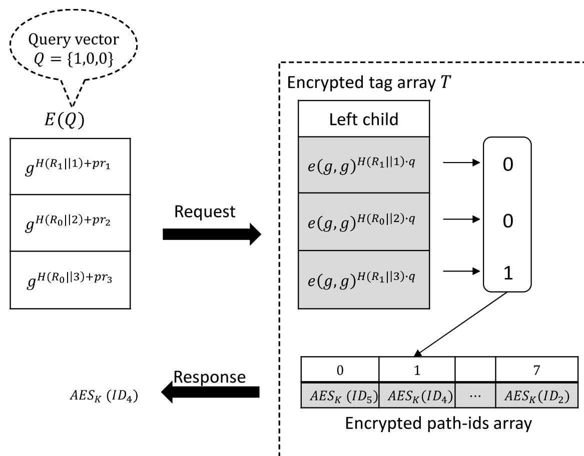


FIGURE 4. An example of query, where the query vector is {1, 0, 0} and $AES_K(ID_4)$ is supposed to be returned.

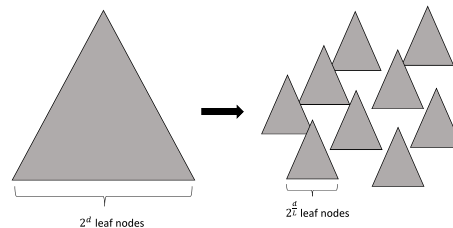


FIGURE 5. An example of dictionary division, where a tree with d height is divided to a forest with L sub-trees of d/L height.

- *Wildcard search:* The wildcard search algorithm extracts all the locations of wildcard keywords from s. For each location i, it copies all the elements in the set $\overline{\mathbb{P}}$ as a set of backup elements, modifies the i-th location value of these backup elements from 0 to 1, and adds them back to the $\overline{\mathbb{P}}$ (lines 12-17).

Fig. 4 depicts an example of query, where the query vector is $\{1, 0, 0\}$ and $AES_K(ID_4)$ is supposed to be returned. In this example, the $\overline{\mathbb{P}}$ outputted from the search algorithm 1 only contains one path vector, i.e., $(0, 0, 1)$, because there is no $*$ in the query vector.

Step 4: According to the path vectors in $\overline{\mathbb{P}}$, the cloud server extracts the corresponding elements from the encrypted path-ids array and returns these elements to the data user, who runs AES decryption algorithm to get the identifiers of matching files. As shown in Fig. 4, the cloud server extracts 2-th element from encrypted path-ids array, i.e., $AES_K(ID_4)$, since the path vector in $\overline{\mathbb{P}}$ is $(0, 0, 1)$.

Step 5: Finally, the data user requests the corresponding encrypted files from the cloud server according to the identifiers.

4) FILE UPDATE

Given a file f_u with a set of keywords $\mathcal{W}_u$ and a file identifier id_u, the data user updates (inserts/deletes) it to the cloud server in the following steps:

Step 1: The data user transforms $\mathcal{W}_u$ to a $1 \times d$ file vector V_u through the basic keyword-vector transformation method. Then, in order to protect the privacy of the update, the data user randomly inserts λ wildcards (i.e., $*$) to the V_u, where the λ is a redundancy factor chosen according to the security level of the system.

Step 2: The data user treats the vector V_u, generated from the last step, as a query vector Q, and performs a conjunctive query with the cloud server. After that, the data user will receive 2^λ encrypted identifier sets from the cloud server and then decrypt them. Next, the data user updates(inserts/deletes) id_u to the identifier set whose path vector is equal to V_u. Finally, the data user re-encrypts all these identifier sets and sends them to the cloud server.

Step 3: If the update operation is insertion, the data user needs to encrypt the file f_u to $AES_k(f_u)$ and sends the $AES_k(f_u)$ to the cloud server.

Note. Since the size of the encrypted path-ids array increases exponentially with the size of keyword dictionary (i.e., d), the computation and storage cost of the proposed scheme will be huge if the d is relatively large. In order to solve this problem, the data user can divide the keyword dictionary into sub-groups by their categories and use the proposed scheme separately on each sub-group. For the example in Fig. 5, the data user divides the keyword dictionary $\mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_d\}$ into L sub-groups: $\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_L$. For each sub-group, the data user treats it as a separate keyword dictionary and applies the proposed scheme on it to get an independent index. During a query, the cloud server conducts search algorithm for each index respectively, and returns L encrypted file identifier sets to the data user, who will decrypt them and calculates the intersection of these file identifier sets to get the matching results. In this way, the storage cost can be greatly reduced from $O(2^d)$ to $O(L \cdot 2^{d/L})$, and the computational cost can be reduced from $O(d_q + 2^{d_\omega})$ to $O(d_q + L \cdot 2^{d_\omega/L})$, where d_q is the number of queried keywords and d_ω is the number of wildcard keywords.

V. SECURITY ANALYSIS

In this section, we will analyze the security of the proposed scheme. Specifically, we mainly focus on the privacy of outsourcing data, conjunctive query, and updated file.

A. OUTSOURCING DATA IS PRIVACY-PRESERVING

In the proposed scheme, the outsourcing data consists of a collection of encrypted files and an encrypted index. For the encrypted files, they are encrypted by the AES encryption (AES-CBC mode) and the security of AES guarantees that they are privacy-preserving. In the following, we mainly analyze the privacy of the encrypted index.

The encrypted index includes two data structures, i.e., an encrypted path-ids array and an encrypted tag array. For the encrypted path-ids array, each element in it is encrypted by the AES encryption (AES-CBC mode). Therefore, the security of AES encryption guarantees it is privacy-preserving.

For the encrypted tag array, as described in subsection IV-B, it is transformed from our conjunctive tree and contains d encrypted elements (only left column), denoted as $\{e(g, g)^{H(R_{t_i}||i)q} | i = 1, 2, \cdots, d\}$, where $t_i \in \{0, 1\}$ represents the permutation rule of i-th level of our conjunctive tree. In the following, we give Theorem 1 to show that the cloud server cannot get any information about the t_i.

Theorem 1: If H is a secure cryptographic hash function, the cloud server cannot get any information about t_i ($1 \leq i \leq d$) from the encrypted tag array.

Proof: We prove the theorem from the following two aspects. First, we consider an encrypted element $e(g, g)^{H(R_{t_i}||i)q}$ individually. On the one hand, although $e(g, g)^q$ can be calculated by the cloud server, the cloud server cannot directly calculate the R_{t_i} from the $e(g, g)^{H(R_{t_i}||i)q}$ because H is a one-way function. On the other hand, the cloud server cannot exhaust all possible R_{t_i} to make sure the value of R_{t_i} because the R_0, R_1 are randomly chosen from Z_N and N is a very large number (1024-bit). Thus, the cloud server cannot get any information about R_{t_i}. Then, it only has 1/2 probability to correctly guess $t_i = 0$ or $t_i = 1$. Second, we consider the relation between different encrypted elements. For each encrypted elements $e(g, g)^{H(R_{t_i}||i)q}$, R_{t_i} is concatenated with a unique level number i before being encrypted, which means all the encrypted elements have different hash inputs (i.e., $R_{t_i}||i$) and unlinkable hash outputs (i.e., $H(R_{t_i}||i)$). Thus, given any two encrypted elements $e(g, g)^{H(R_{t_{i_1}}||i_1)q}$ and $e(g, g)^{H(R_{t_{i_2}}||i_2)q}$, the cloud server cannot know any information about the relation between $R_{t_{i_1}}$ and $R_{t_{i_1}}$, i.e., $t_{i_1} = t_{i_2}$ or $t_{i_1} \neq t_{i_2}$.

Therefore, the cloud server cannot get any information about t_i ($1 \leq i \leq d$) from the encrypted tag array. □

Based on the Theorem 1, we also can obtain that the cloud server cannot get any information about the permutation rules of the conjunctive tree since each t_i ($1 \leq i \leq d$) represents the permutation rule of i-th level of our conjunctive tree. Furthermore, given any leaf in the encrypted conjunctive tree, the cloud server only has $1/2^d$ probability to correctly guess its original location. In other words, our conjunctive tree is privacy-preserving.

B. CONJUNCTIVE QUERY IS PRIVACY-PRESERVING

The conjunctive query consists of three steps: query request, query processing and query response. We consider different attackers in different steps: for the query request and query response, we focus on the attack from the outside entity, who tries to determine whether two queries have the same queried keywords; for the query processing, we focus on the attack from the cloud server, who is curious about the content of queries. In the following, we will show that these steps are all privacy-preserving.

In the query request, the data user encrypts a query vector $Q = (k_0, k_1, \ldots, k_{d-1})$ to an encrypted query vector $E(Q) = (E(k_0), E(k_1), \ldots, E(k_{d-1}))$. Specifically, for each $k_i \in Q$, the data user encrypts it to $E(k_i) = g^{H(R_{k_i}||i)+pr_i}$ (see Eq. (2)), which guarantees k_i is secure. Meanwhile, since the data user selects a random value r_i for each query, the above encryption is non-deterministic. In other word, for the same query vector Q, the ciphertext $E(Q)$ will be different at different runs. Therefore, the outside entity cannot determine whether two queries have the same queried keywords.

In the query processing, after receiving the encrypted query vector $E(Q) = (E(k_0), E(k_1), \ldots, E(k_{d-1}))$, the cloud server computes $e(E(k_i), h)$ and compares it with the i-th element in the encrypted tag array for each i. Then, the cloud server can get the queried permuted path vectors for this query. Since the permuted path vectors will not expose any information of path vectors, we can see that the query processing will not leak any information except the access pattern (i.e., search result) and search pattern (i.e., which queries have the same queried keywords) to the cloud server.

In the query response, the cloud server responds the queried encrypted file identifier sets, which is encrypted by AES encryption (AES-CBC mode), to the data user. Since the AES-CBC mode can be non-deterministic, e.g., by inserting a random number as the first block, the outside entity cannot determine whether two queries have the same queried keywords.

C. UPDATED FILE IS PRIVACY-PRESERVING

In the update, given a file identifier id_u and a file vector v_u, the data user first randomly inserts λ wildcards, i.e., redundancy, to the v_u and performs a multi-keyword conjunctive query request to retrieve 2^λ encrypted identifier sets. Then, the data user decrypts all these encrypted file identifier sets, updates one of them (i.e., ID_u), and re-encrypts them through AES encryption (AES-CBC mode). Since the AES-CBC mode can be non-deterministic, e.g., by inserting a random number in the first block, the cloud server cannot distinguish which identifier set has been modified. As a result, the updated file is privacy-preserving. Note that, this privacy depends on the size of redundancy factor λ, which thus leads to a tradeoff between the security and communication overhead.

VI. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our proposed scheme from both theoretical and experimental perspectives.

A. THEORETICAL ANALYSIS

In this subsection, we theoretically analyze the performance of our proposed scheme. Since our design goal is to improve the query efficiency, we focus on analyzing the computational cost and communication overhead of the query. At the same time, we compare it with some traditional algorithms.

In our proposed scheme, the search algorithm includes two steps: basic search and wildcard search (see Alg. 1). Assume d is the size of keyword dictionary, d_q is the number of queried keywords, and $d_\omega = d - d_q$ is the number of wildcard keywords. Then the basic search will cost d_q pairing operations, and the wildcard search will cost 2^{d_ω} string copy

TABLE 2. Comparisons between ours and existing schemes.

Scheme	Query Computation	Query Communication
[13]	$O(n \cdot d)$	$O(d + n_q)$
[8]	$O(n_{\omega_1} \cdot d_q)$	$O(d_q + n_q)$
[10]	$O(d^2)$	$O(d + n_q)$
[14]	$O(n_{\omega_1} \cdot d_q)$	$O(d_q + n_q)$
Our scheme	$O(d_q)$	$O(d_q + n_q)$

n is the size of file collection, n_q is the size of retrieval files, d is the size of keyword dictionary, d_q is the size of queried keyword, n_{ω_1} is the size of retrieval files for the first queried keyword ω_1.

operations. Since we consider relatively small d_ω and the cost of a string copy operation is very small, the 2^{d_ω} string copy operations can be ignored. Therefore, our query computational cost is approximately equal to $O(d_q)$, which is significantly more efficient than [8], [10], [13], [14] according to the TABLE 2. Meanwhile, during a query, the communication overhead in our scheme is $O(d_q + n_q)$, where $O(d_q)$ is from the query request and $O(n_q)$ is from the query response. From the TABLE 2, we can see that this communication overhead is already optimal in the existing schemes.

B. EXPERIMENTAL ANALYSIS

In this subsection, we experimentally analyze the computational cost of our proposed scheme. Since we have theoretically compared our solution with others in the last subsection, we just implement our solution here.

We evaluate the computational cost of our proposed scheme in terms of three phases: local data outsourcing, conjunctive query and file update. Specifically, we implement the proposed scheme in C/C++ (our code is open source [15]) and conduct experiments on a 64-bit machine with an Intel(R) Core(TM) i5-4300M CPU at 2.6GHZ and 4GB RAM, running CentOS 6.6. We utilize the OpenSSL and PBC library for the entailed cryptographic operations: the BGN parameters are generated through the type a1 pairing in PBC library, which is constructed on the curve $y^2 = x^3 + x$ (the group order N is a 1024-bit number); the hash function $H(\cdot)$ and AES are instantiated using SHA-512 and AES-512-CBC in the OpenSSL library respectively. Note that, we implement the data user and the cloud server on the same machine, which means there is no network delay between them. In addition, our experiments are based on different synthetic file collections, which consist of $1 \times d$ binary vectors (i.e., file vectors) chosen randomly. The size of file collections is from 10000 to 60000, and the size of keyword dictionary d is from 5 to 10. In the following, we will show our experiment results and analyze them.

1) LOCAL DATA OUTSOURCING

First, we evaluate the computational cost of local data outsourcing. In our proposed scheme, the computational cost of local data outsourcing is mainly from two parts: building the encrypted tag array and building the encrypted path-ids array.

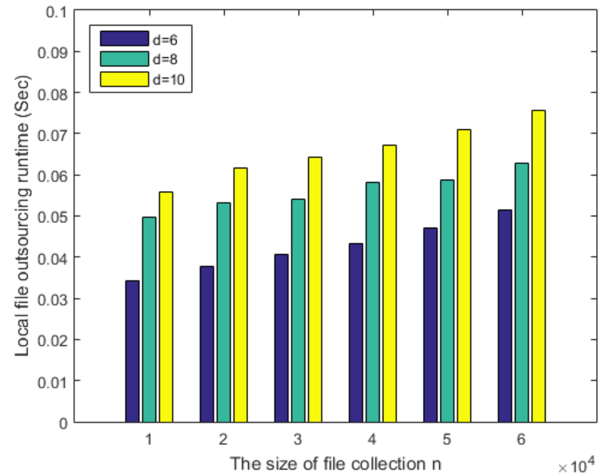


FIGURE 6. Local file outsourcing runtime versus the size of file collection n without division, where $d \in \{6, 8, 10\}$ and $L = 1$.

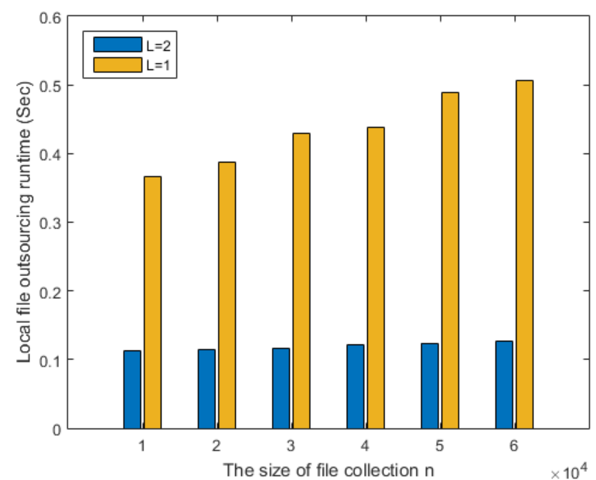


FIGURE 7. Local file outsourcing runtime versus the size of file collection n with division, where $d = 20$ and $L \in \{1, 2\}$.

To be more specific, we assume the size of file collection is n and keyword dictionary is d. Then the first part will cost d modular exponentiations in group G_T and the second part will cost 2^d AES encryptions. As shown in Fig. 6, the runtime of local data outsourcing versus the size of file collection (i.e., n) is plotted, where the keyword dictionary $d \in \{6, 8, 10\}$. From this figure, we can see that the computational cost increases linearly with n and exponentially with d.

When the d is relatively large, the proposed scheme would be impractical because of the 2^d AES encryptions. To solve this problem, the data user can divide keyword dictionary into L sub-groups and apply the proposed scheme on each subgroup separately. In this way, the computational cost of AES encryptions can be reduced from 2^d to $L \cdot 2^{d/L}$. Fig. 7 depicts the local data outsourcing runtime versus the size of file collection with different divisional size L, where $d = 20$ and $L \in \{1, 2\}$. This figure shows that this exponential growth of computational cost can be effectively limited by the division.

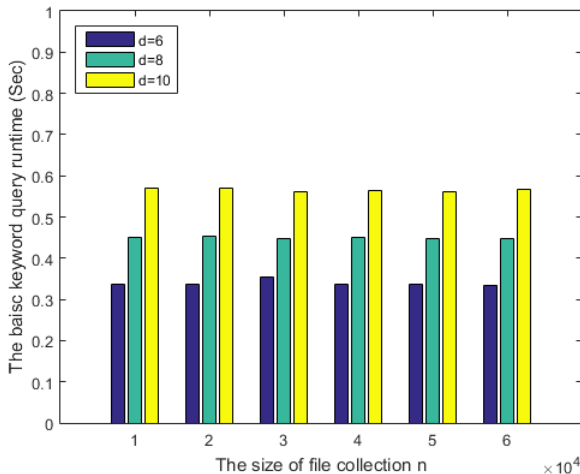


FIGURE 8. Basic keyword query runtime versus the size of file collection n, where $6 \le d \le 10$.

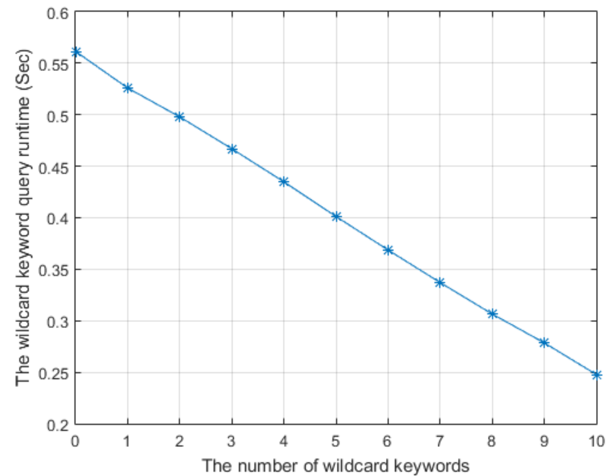


FIGURE 9. Wildcard keyword query runtime versus the number of wildcard keywords, where $d = 10$ and $n = 6 \times 10^4$.

2) CONJUNCTIVE QUERY

Then, we evaluate the computational cost on two types of keyword queries: basic query and wildcard keyword query. For the former, there is no wildcard keyword included in the query so that the cloud server only needs to find the first matching permuted path vector, which only requires d pairing operations. For the latter, there are multiple permuted path vectors satisfying the query. Assume the number of wildcard keywords in the query is d_w, then the cloud server needs to perform $d - d_w$ pairing operations for the first matching permuted path vector and 2^{d_w} string copy operations for the rest.

In Fig. 8, the runtime of basic keyword query versus the size of file collection is plotted. From this figure, we can see that the computational cost is indeed linearly increasing to d, but not affected by the n.

Fig. 9 depicts the runtime of wildcard keyword query versus the number of wildcard keywords (i.e., d_w). In this evaluation, we use the fixed size of keyword dictionary ($d = 10$) and file collection ($n = 6 \times 10^4$). From this figure, we can see that the runtime decreases as the d_w increases because the computational cost of a pairing operation is much larger than a string copy operation. In order to clarify this, we let T_e and T_c denote the computational cost of a pairing operation and a string copy operation respectively. Then the computational cost of a wildcard keyword query is $(d-d_w)\cdot T_e+2^{d_w}\cdot T_c$. This formula shows that when the d_w is relative small, we can get $d_w \cdot T_e > 2^{d_w} \cdot T_c$, which means the increasing d_w will reduce the computational cost.

3) UPDATE

Finally, we consider the computational cost on the update. According to the previous description (see IV-B.4), the data user adds redundancies in the update request to protect the privacy. In this way, the update can be seen as a wildcard keyword query, whose query vector has λ wildcard keywords. Therefore, the runtime of update(insertion/deletion) is the same as the wildcard keyword query with $d_w = \lambda$.

VII. RELATED WORK

In the single user/single server system, a searchable encryption scheme can be realized with optimal security via powerful cryptographic tools, such as Fully Homomorphic Encryption (FHE) [2], [16] and Oblivious Random Access Memory (ORAM) [3], [17]. However, these tools are extraordinary impractical. Another set of works utilize property-preserving encryption (PPE) [4], [18]–[20], which encrypts messages in a way that inevitably leaks certain properties of the underlying message. For balancing the leakage and efficiency, many studies focus on Searchable Symmetric Encryption (SSE). Song *et al.* [21] first used the symmetric encryption to facilitate keyword search over the encrypted data. Then, Curtmola *et al.* [22] gave a formal definition of SSE, and proposed an efficient SSE scheme. However, their scheme cannot support update(insertion/deletion). Later, Kamara *et al.* [23] proposed the first dynamic SSE scheme, which uses a deletion array and a homomorphic encrypted pointer technique to securely update files. Unfortunately, due to the use of fully homomorphic encryption, the update efficiency is very low. In a more recent paper [24], Cash et al. described a simple dynamic inverted index based on [22], which utilizes the data unlinkability of hash table to achieve secure insertion. However, this paper uses a revocation list to implement deletion instead of actually removing a record from the inverted index, which will cause the storage and computation overhead continue to increase.

Our work follows the previous works but focuses on the multi-keyword conjunctive query. In the rest of this section, we briefly review some recently proposed privacy-preserving multi-keyword conjunctive query schemes. Golle *et al.* [13] proposed the first scheme to support conjunctive queries. In this scheme, they build a forward index for each file, which means the cloud server needs to traverse all the forward indexes during search. Therefore, the query computational cost of this scheme is linear in the number of files. In [10], Bing et al. proposed a scheme which uses a private set

intersection (PSI) technique (based on Paillier encryption) to solve the conjunctive query problem. However, the computational cost of query is $O(d^2)$, where d is the size of the keyword dictionary. In [8] and [14], the authors used an Oblivious Cross-Tag (OXT) protocol to support conjunctive queries. This protocol builds two indexes, one of which is an inverted index TSet and another is a forward index XSet. The TSet is used to search for a single keyword, and the XSet is used to filter out unnecessary file identifiers from TSet such that only those contain all the keywords are returned. The OXT protocol can be extended to process any form of boolean query, but it takes time linear in the number of files to search in the worst case. Base on the [8], Hu *et al.* [9] used function-hiding inner product encryption (IPE) [25] to implement forward indexes, which can achieve forward security. Unfortunately, the search efficiency is still linear in the number of files.

Different from the above works, our proposed scheme can achieve approximate $O(d_q)$ computation efficiency for the multi-keyword conjunctive query (see TABLE 2), where d_q is the number of queried keywords in the query. Besides, the proposed scheme also supports NOT operation for queried keywords, which is difficult for the prior works.

VIII. CONCLUSION

In this paper, we have proposed a novel efficient and privacy-preserving multi-keyword conjunctive query scheme. Specifically, we designed a tree-based index to support multi-keyword conjunctive query, and a wildcard search method to speed up the query processing for wildcard keywords. In addition, we utilize the permutation and BGN algorithm to hide the order information (i.e., path vector) leaked from the tree-based index. Security analysis shows the proposed scheme is privacy-preserving and performance evaluation also validates its efficiency. In the future work, we will take more security properties into consideration, e.g., forward security and backward security. Furthermore, we will study the efficient problem of the fuzzy query.

REFERENCES

[1] I. M. Cloud. (2010). *Key Marketing Trends for 2017 and Ideas for Exceeding Customer Expectations*. [Online]. Available: https://bizibl.com/marketing/download/10-key-marketing-trends-2017-and-ideas-exceeding-customer-expectations

[2] C. Gentry, ''Fully homomorphic encryption using ideal lattices,'' in *Proc. 41st Annu. ACM Symp. Theory Comput. (STOC)*, Bethesda, MD, USA, May/Jun. 2009, pp. 169–178.

[3] R. Ostrovsky, ''Efficient computation on oblivious RAMs,'' in *Proc. 22nd Annu. ACM Symp. Theory Comput.*, Baltimore, MD, USA, May 1990, pp. 514–523.

[4] M. Bellare, A. Boldyreva, and A. O'Neill, ''Deterministic and efficiently searchable encryption,'' in *Proc. Annu. Int. Cryptol. Conf. (CRYPTO)*, Santa Barbara, CA, USA, Aug. 2007, pp. 535–552.

[5] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. N. Garofalakis, ''Practical private range search revisited,'' in *Proc. Int. Conf. Manage. Data (SIGMOD)*, San Francisco, CA, USA, Jun./Jul. 2016, pp. 185–198.

[6] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias, ''Delegatable pseudorandom functions and applications,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Berlin, Germany, Nov. 2013, pp. 669–684.

[7] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, ''Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security,'' in *Proc. Eur. Symp. Res. Comput. Secur. (ESORICS)*, Barcelona, Spain, Sep. 2018, pp. 228–246.

[8] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, ''Highly-scalable searchable symmetric encryption with support for Boolean queries,'' in *Proc. Annu. Cryptol. Conf. (CRYPTO)*, Santa Barbara, CA, USA, Aug. 2013, pp. 353–373.

[9] C. Hu, X. Song, P. Liu, Y. Xin, Y. Xu, Y. Duan, and R. Hao, ''Forward secure conjunctive-keyword searchable encryption,'' *IEEE Access*, vol. 7, pp. 35035–35048, 2019.

[10] B. Wang, W. Song, W. Lou, and Y. T. Hou, ''Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee,'' in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Hong Kong, Apr./May 2015, pp. 2092–2100.

[11] D. Boneh, E.-J. Goh, and K. Nissim, ''Evaluating 2-DNF formulas on ciphertexts,'' in *Proc. Theory Cryptogr. Conf. (TCC)*, Cambridge, MA, USA, Feb. 2005, pp. 325–341.

[12] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.

[13] P. Golle, J. Staddon, and B. R. Waters, ''Secure conjunctive keyword search over encrypted data,'' in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur. (ACNS)*, Huangshan, China, Jun. 2004, pp. 31–45.

[14] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, D. Mukhopadhyay, R. Steinfeld, S.-F. Sun, D. Liu, and C. Zuo, ''Result pattern hiding searchable encryption for conjunctive queries,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, Oct. 2018, pp. 745–762.

[15] F. Yin. (2019). *An Implementation of Our Proposed Scheme*. [Online]. Available: https://github.com/YinFFF/Multi-keyword-SSE

[16] C. Gentry, ''Computing arbitrary functions of encrypted data,'' *Commun. ACM*, vol. 53, no. 3, pp. 97–105, Mar. 2010.

[17] O. Goldreich and R. Ostrovsky, ''Software protection and simulation on oblivious RAMs,'' *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[18] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, ''Order-preserving symmetric encryption,'' in *Proc. 28th Annu. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*, Cologne, Germany, Apr. 2009, pp. 224–241.

[19] A. Boldyreva, N. Chenette, and A. O'Neill, ''Order-preserving encryption revisited: Improved security analysis and alternative solutions,'' in *Proc. Annu. Cryptol. Conf. (CRYPTO)*, Santa Barbara, CA, USA, Aug. 2011, pp. 578–595.

[20] W. Yang, Y. Xu, Y. Nie, Y. Shen, and L. Huang, ''TRQED: Secure and fast tree-based private range queries over encrypted cloud,'' in *Proc. Int. Conf. Database Syst. Adv. Appl. (DASFAA)*, Gold Coast, QLD, Australia, May 2018, pp. 130–146.

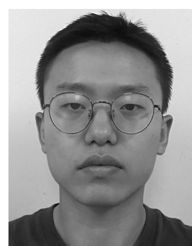
[21] D. X. Song, D. Wagner, and A. Perrig, ''Practical techniques for searches on encrypted data,'' in *Proc. IEEE Symp. Secur. Privacy*, Berkeley, CA, USA, May 2000, pp. 44–55.

[22] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, ''Searchable symmetric encryption: Improved definitions and efficient constructions,'' in *Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS)*, Alexandria, VA, USA, Oct./Nov. 2006, pp. 79–88.

[23] S. Kamara, C. Papamanthou, and T. Roeder, ''Dynamic searchable symmetric encryption,'' in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, Raleigh, NC, USA, Oct. 2012, pp. 965–976.

[24] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, ''Dynamic searchable encryption in very-large databases: Data structures and implementation,'' in *Proc. NDSS*, vol. 14, 2014, pp. 23–26.

[25] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu, ''Function-hiding inner product encryption is practical,'' in *Proc. Int. Conf. Secur. Cryptogr. Netw. (SCN)*, Amalfi, Italy, Sep. 2018, pp. 544–562.



FAN YIN received the B.S. degree in information security from Southwest Jiaotong University, Chengdu, China, in 2012, where he is currently pursuing the Ph.D. degree in information and communication engineering. He is also a Visiting Student with the Faculty of Computer Science, University of New Brunswick, Canada. His research interests include searchable encryption, privacy-preserving, and security for cloud security and network security.

YANDONG ZHENG received the M.S. degree from the Department of Computer Science, Beihang University, China, in 2017. She is currently pursuing the Ph.D. degree with the Faculty of Computer Science, University of New Brunswick, Canada. Her research interests include cloud computing security, big data privacy, and applied privacy.

RONGXING LU (S'09–M'11–SM'15) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2012. He worked as a Postdoctoral Fellow with the University of Waterloo, from May 2012 to April 2013. He worked also as an Assistant Professor with the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore, from April 2013 to August 2016. He is currently an Associate Professor with the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. His research interests include applied cryptography, privacy enhancing technologies, and the IoT-big data security and privacy. He has published extensively in his areas of expertise. Dr. Lu was a recipient of eight best (student) paper awards from some reputable journals and conferences. He was awarded the most prestigious Governor General's Gold Medal, while doing his Ph.D. degree, and also received the 8th IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award, in 2013. He is a Senior Member of the IEEE Communications Society. He currently serves as the Vice-Chair (Publication) of the IEEE ComSoc CIS-TC (Communications and Information Security Technical Committee). He is the Winner of 2016–2017 Excellence in Teaching Award, FCS, and UNB.

XIAOHU TANG (M'04–SM'18) received the B.S. degree in applied mathematics from Northwest Polytechnic University, Xi'an, China, in 1992, the M.S. degree in applied mathematics from Sichuan University, Chengdu, China, in 1995, and the Ph.D. degree in electronic engineering from Southwest Jiaotong University, Chengdu, in 2001.

From 2003 to 2004, he was a Research Associate with the Department of Electrical and Electronic Engineering, The Hong Kong University of Science and Technology. From 2007 to 2008, he was a Visiting Professor with the University of Ulm, Germany. Since 2001, he has been with the School of Information Science and Technology, Southwest Jiaotong University, where he is currently a Professor. His research interests include coding theory, network security, distributed storage, and information processing for big data.

Dr. Tang was a recipient of the National excellent Doctoral Dissertation Award, China, in 2003, the Humboldt Research Fellowship, Germany, in 2007, and the Outstanding Young Scientist Award by NSFC, China, in 2013. He served as an Associate Editor for several journals, including the IEEE Transactions on Information Theory and the *IEICE Transactions on Fundamentals*, and served for a number of technical program committees of conferences.