Hindawi
Security and Communication Networks
Volume 2021, Article ID 9023141, 13 pages
https://doi.org/10.1155/2021/9023141

Research Article

Attribute Set-Based Boolean Keyword Search over Encrypted Personal Health Records

Yu Lin,[1] **Lingling Xu**,[1] **Wanhua Li**,[1] **and Zhiwei Sun**[2]

[1]*School of Computer Science and Engineering, South China University of Technology, Guangzhou, China*
[2]*School of Artificial Intelligence, Shenzhen Polytechnic, Shenzhen, China*

Correspondence should be addressed to Lingling Xu; csllxu@scut.edu.cn

Received 24 September 2021; Accepted 11 November 2021; Published 23 December 2021

Academic Editor: Xin-Yi Huang

A personal health record (PHR) is an electronic application which enables patients to collect and share their health information. With the development of cloud computing, many PHR services have been outsourced to cloud servers. Cloud computing makes it easier for patients to manage their personal health records and makes it easier for doctors and researchers to share and access this information. However, due to the high sensitivity of PHR, a series of security protections are needed to protect them, such as encryption and access control. In this article, we propose an attribute set-based Boolean keyword search scheme, which can realize fine-grained access control and Boolean keyword search over encrypted PHR. Compared with the existing attribute-based searchable encryption, our solution can not only improve the flexibility in specifying access policies but also perform Boolean keyword search, which can meet the needs of large-scale PHR users. Furthermore, we simulate our scheme, and the experimental results show that our scheme is practical for PHR systems in cloud computing.

1. Introduction

In recent years, with the rapid development of cloud computing, it has been successfully deployed in a wide variety of real-world applications, such as the medical industry. A personal health record (PHR) is an electronic application through which patients can maintain and manage their health information in a private, secure, and confidential environment. The intention of PHR is to provide a complete and accurate summary of an individual's medical history which is accessible online. PHR benefits in many aspects, including strengthening the connection between doctors, patients, medical researchers, and hospitals. Doctors and medical researchers can obtain patient's information more conveniently by using PHR. At the same time, patients can better control their personal health information (PHI) since only people with necessary electronic credentials can get access to their PHR. However, PHR is under security threat such as information leakage, lack of access control, and untrusted cloud servers. A lot of security issues have happened in recent years, which have brought huge economic losses. Thus, it is very important to develop a method for protecting the security and privacy of PHR. The most common method is to encrypt the PHR before uploading them to the server. However, performing searches on encrypted PHR would be a challenge.

Searchable encryption (SE) is a well-studied method that can deal with this issue. In SE, firstly, data owners encrypt their data and upload them to the cloud server. Then, each legitimate data user can generate a trapdoor using his secret key and interested keywords where the trapdoor enables data user to search over encrypted PHR. However, due to the high sensitivity, it is desired that each user can only query the authorized PHR. That is, fine-grained access control is needed for PHR. Thus, attribute-based keyword search has been implemented by many researchers [1–12]. In these schemes, data owners can encrypt their data with predefined access control policies. A trusted authority is responsible for managing all data users and distributing secret keys to them, where the secret key is generated according to the user's attributes. Thus, when a data user searches over encrypted PHR stored in the server with his interested keywords, the server can judge whether his attributes satisfy the access policy.

However, there are some limitations in existing schemes. Firstly, in the schemes [7–10], each user's attributes are organized in a single set, which cannot support the compounded attributes. For example, consider a user who is a "Researcher" working in "College A" and serves as both "Director" of "Department of Medicine" and "Professor" of "Department of Chemistry," and the above attributes are both valid and are likely to be used to describe him. A possible method is expressing above attributes as strings, like "Researcher_College *A*_Director_ Department Of *M*–edicine_Professor_ Department Of Chemistry". Unfortunately, it becomes challenging in satisfying policies which combine some of the singleton attributes. Secondly, in the schemes [10–12], data users are only allowed to search with a single keyword, which is not flexible enough. Although the scheme [7] supports recursive attribute set structure for more flexibility in specifying policies and the scheme [3] supports Boolean keyword search, none of the schemes support both of the two functionalities.

1.1. Our Contributions. In this paper, we present an attribute set-based Boolean keyword search (ASBBKS) scheme over encrypted PHR in cloud computing. In our scheme, each data user's attributes are organized as recursive set structure, which enables more flexibility in user attribute organization and more efficiency in specifying policies than the existing ABKS schemes. Meanwhile, our scheme supports Boolean keyword search which is flexible in keyword expressivity. The ASBBKS model for encrypted PHR is shown in Figure 1. The main contributions of our paper are described in detail as follows:

(i) In our scheme, each data user's attributes can be organized as recursive set structure. That is, multiple values are assigned to an attribute in different sets. In the above example, the researcher's attributes can be organized into a 2-depth recursive set structure as follows. For each role that a researcher has, a separate set of values {Department, Role} can be assigned.

{Agency: College A, Role: Researcher,

{Department: Medicine, Role: Director},

{Department: Chemistry, Role: Professor}}.

(ii) All authorized users can perform Boolean keyword search which is a more flexible search mechanism. For example, the PHR users can query the PHR which contains the keywords "(Treatment time: Jun 2021 $\wedge$ Doctor name: Mike) $\wedge$ (Drug name: Aspirin $\vee$ Disease name: Neuralgia)" to the cloud server.

(iii) We present the theoretical performance analysis of the ASBBKS scheme for computation and communication costs. In addition, the simulated results show that it is practical for the PHR systems.

1.2. Related Works. Searchable encryption (SE) was first proposed by Sont et al. [13] which enables users to implement keyword research over the encrypted data. There are two categories for existing SE schemes: symmetric searchable encryption (SSE) [13–17] and public key encryption with keyword search (PEKS) [18–21]. SSE enables a user to encrypt his data and implement the keyword search by using his secret key. In a PEKS scheme, data owners can authorize search ability to each user by encrypting data with the user's public key. Thus, each user can perform searches over encrypted data with his private key. However, the above two types of SE have a vulnerability that they do not support access control on data users. It means that each legitimate user can query all of the encrypted data, which may cause security issues.

To solve this problem, researchers have proposed SE schemes with access control such as attribute-based encryption with keyword search (ABKS) [22–26]. In such SE schemes, data owners can authorize search ability to data users whose attributes satisfy the access policy predefined by the data owner. There is a trusted authority in ABKS to generate public parameters. In addition, it also generates secret keys for data users. In [27], each user needs to submit his attribute structure and interested keywords to the trusted authority for generating trapdoors. Then, the data user can search over encrypted data with his trapdoors. However, it fails to preserve the privacy of interested keywords and they will be revealed to the trusted authority, which is not desired by the data user. Thus, Zheng et al. [28] proposed an improved SE scheme that makes use of an access tree to fulfill access control. In [28], each user sends an attribute set to the trusted authority for generating an attribute-related private key. The private key is used to generate trapdoors of interested keywords. Then, users can search over encrypted data with trapdoors without revealing interested keywords. However, in these schemes, user's attributes are organized to a single set, which decreases the flexibility in specifying access policies.

In [29], the researchers proposed attribute set-based encryption (ASBE). In ASBE, the user's attributes are organized in the form of recursive set structure which enables the data owners to encrypt data with more flexible access policies than ABE. Based on this technique, Xu et al. [7] proposed attribute set-based keyword search (ASBKS). Their scheme has better flexibility in user attribute organization and more efficiency in specifying policies. However, it only supports single keyword query, which limits the performance of searching.

To achieve multi-keyword search, Boolean keyword expression search was presented in [1]. In [1], keywords are divided into two parts: value and name, which are both organized in the form of an access tree structure. Inspired by this, we propose an ASBBKS scheme that can support Boolean keyword search, compounded attributes, and flexible search policies simultaneously.

1.3. Paper Organization. The rest of the paper is organized as follows. We introduce some preliminaries in Section 2. In Section 3, the concrete construction of the ASBBKS scheme is presented and the formal security proof of the ASBBKS scheme is provided. In Section 4, we evaluate the

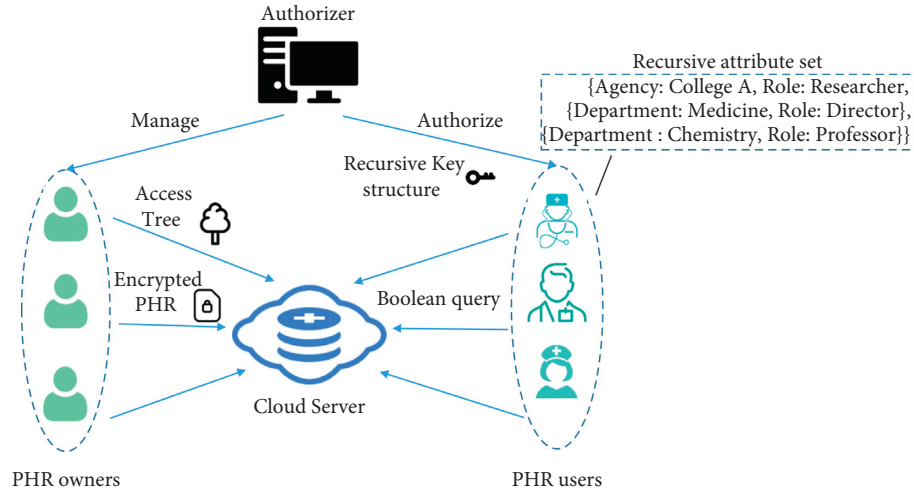

Figure 1: Our scheme model.

performance of our ASBBKS scheme and conduct simulation experiments to demonstrate its efficiency and practicality. Finally, we conclude the paper in Section 5.

2. Preliminaries

2.1. Bilinear Pairings. Let G_1 and G_2 represent the multiplicative cyclic group of order p, where p represents the prime number. Assume that g is the generator of group G_1; then, when e: $G_1 \times G_1 \longrightarrow G_2$ satisfies the following conditions, we say it is a bilinear mapping [30]:

(1) Bilinear: for any $g, h \in G_1$ and $a, b \in Z_p$, $e(g^a, h^b) = e(g, h)^{ab}$.

(2) Non-degenerate: $e(g, g) \neq 1$.

If there exists an efficient algorithm that can compute $e(g, h)$ for all $g, h \in G_1$, then G_1 is called a bilinear group.

2.2. Recursive Set Structure. We construct user's attributes as the recursive structure mentioned in [29]. In this structure, each element of a set can be an associated attribute or a set. They are organized like the tree structure with the notation of depth which limits this recursion. Here, we provide an example of this kind of set structure with depth 2 in Figure 2.

At the first layer, there are attribute elements and set elements. At the second layer, there are only attribute elements. For a set with depth 2, we can denote it as $\mathbb{A} = \{A_0, A_1, \ldots, A_n\}$ where A_0 is the set of attributes at depth 1 and A_i is the ith set at depth 2 for $1 \leqslant i \leqslant n$.

2.3. Access Tree. In our scheme, the access tree structure is the same as that in [29], and Figure 3 shows an example of access tree $\mathcal{T}$. In $\mathcal{T}$, each inner node x represents a threshold gate which has a threshold value k_x. Assume that the set of child nodes of x is represented by child(x) and the number of child nodes is represented by n_x. In addition, each child node of x is labeled from left to right as 1 to n_x. Then, we have $1 \leq k_x \leq n_x$. When $k_x = 1$, the threshold gate transforms to a " OR " gate; when $k_x = n_x$, it transforms to an " AND " gate. In addition, each leaf node x of $\mathcal{T}$ represents an attribute, denoted as att(x). Furthermore, the parent of node x is presented as parent(x), the label associated with x is presented as label(x), the set of leaf nodes of $\mathcal{T}$ is presented as lvs$(\mathcal{T})$, and the subtree of $\mathcal{T}$ rooted at the node x is presented as $\mathcal{T}_x$.

For an attribute set structure $\mathbb{A} = \{A_0, A_1, \ldots, A_n\}$ with depth 2 and an access tree $\mathcal{T}$, if at least one of the following conditions holds, we say that $\mathbb{A}$ satisfies $\mathcal{T}$: (1) there is at least one subset of $\mathbb{A}$, in which the combination of its attributes satisfies $\mathcal{T}$; (2) there are some translating nodes where the attributes from multiple sets in $\mathbb{A}$ can be combined to satisfy $\mathcal{T}$. The translating node allows attributes from different sets to be combined to satisfy an access tree. Thus, by using some translating nodes, data users can combine attributes from multiple sets to satisfy an access tree. To make an explanation, we take the recursive set structure in Figure 2 and $\mathcal{T}$ for example. In access tree $\mathcal{T}$, there are two inner nodes and one of them is a translating node. Their threshold gates are 1 and 2, respectively. For the attribute set structure $\mathbb{A}$ in Figure 2, it can easily satisfy node v_1. For node v_2, subsets A_1 and A_2 can be combined together to satisfy it. However, if node v_2 is not a translating node, then there is no subset of $\mathbb{A}$ that can satisfy node v_2. Thus, using some translating nodes, data owners can selectively require users to combine attributes from either a single set or multiple sets to satisfy the access tree.

Secret share: for an access tree $\mathcal{T}$, each node x of $\mathcal{T}$ with associated threshold k_x would equip with a polynomial q_x.

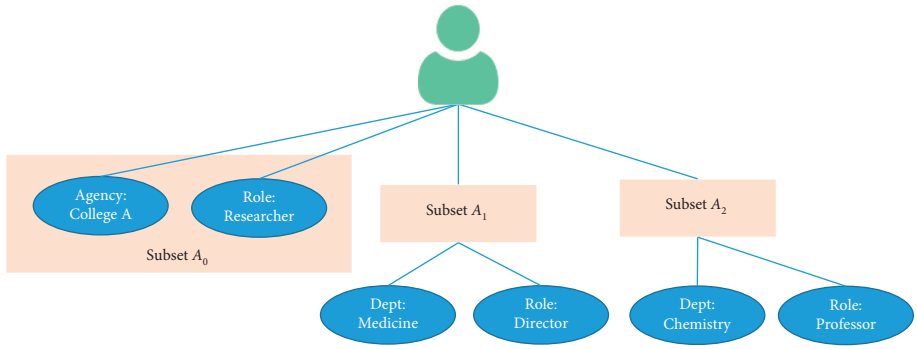

Figure 2: The recursive set structure.

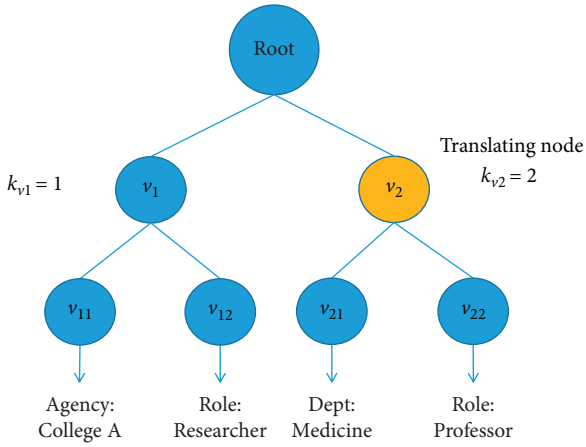

Figure 3: Access tree.

q_x is generated by running the secret share algorithm mentioned in [31]. This algorithm is used for distributing a secret s, and the process of this algorithm is as follows:

(i) For the root node R, let $q_R(0) = s$ and then randomly select $k_R - 1$ coefficients for the polynomial q_R.

(ii) For an inner node x, let $q_x(0) = q_{\text{parent}(x)}(\text{label}(x))$ and then randomly select $k_x - 1$ coefficients for the polynomial q_x.

(iii) For a leaf node x, let the degree of q_x be $d_x = 0$ and $q_x(0) = q_{\text{parent}(x)}(\text{label}(x))$.

After executing the algorithm, the value $q_x(0)$ corresponding to each leaf node x becomes the secret share of s. We represent this algorithm as $\{q_x(0) \mid x \in \text{lvs}(\mathcal{T})\} \longleftarrow \text{Share}(\mathcal{T}, s)$.

2.4. Boolean Query. In [3], a Boolean query requires that a server can find out all the encrypted data associated with an arbitrary Boolean expression among keywords. The expression can be denoted as $\text{Exp}(w_1, w_2, \ldots, w_n)$. While calculating a Boolean expression, let $b_i = 1$ if the encrypted data contains w_i; otherwise, set $b_i = 0$. Then, replace corresponding w_i in the expression with b_i. After that, if the result of the Boolean expression is "1," encrypted data satisfy the condition of the Boolean query. In our scheme, keywords are divided into two parts: name and value. They are organized as an access tree structure as shown in Figure 4, which is a more expressive searchable mechanism.

2.5. Definition of ASBBKS. In an ASBBKS scheme, there are several participants including multiple data owners and data users, a cloud server, and a trusted authority used for authorizing and managing data owners/users. An ASBBKS scheme consists of five algorithms: Setup, KeyGen, Encrypt, Trapdoor, and Test.

(i) $\text{Setup}(k) \longrightarrow \{\text{pk}, \text{mk}\}$: given the security parameter k, the Setup algorithm generates a public key pk and a master key mk.

(ii) $\text{KeyGen}(\text{mk}, \mathbb{A}) \longrightarrow \{\text{sk}\}$: the KeyGen algorithm uses the master key mk and the given attribute set structure $\mathbb{A}$ to generate secret key sk corresponding to $\mathbb{A}$.

(iii) $\text{Encrypt}(\text{pk}, W_V, \mathcal{T}) \longrightarrow \{C\}$: the Encrypt algorithm inputs public key pk, a set of keyword values W_V, and an access tree $\mathcal{T}$ and outputs the corresponding ciphertext C.

(iv) $\text{Trapdoor}(\text{sk}, B_V) \longrightarrow \{\mathcal{T}\}$: the Trapdoor algorithm inputs secret key sk and a Boolean keyword value expression B_v and outputs the corresponding trapdoor T.

(v) $\text{Test}(C, T) \longrightarrow \{0, 1\}$: the Test algorithm takes trapdoor T and ciphertext C as input. When the attribute set related to T matches with the access tree encrypted in C and a minimum subset W'_N of keyword names W_N encrypted in C satisfies the Boolean keyword expression B_V encrypted in T, it outputs 1. Otherwise, it outputs "$\perp$".

In an ASBBKS scheme, the trusted authority firstly executes Setup to initialize the system parameters. Then, according to the user's attribute set, it computes secret key for each user. Data owners should execute Encrypt on their data using preset access control policies to generate the corresponding ciphertexts and then upload them to the server. For data users, they can execute Trapdoor to generate trapdoors for searching over ciphertexts. In the end, the server executes Test to find all the data that are authorized to a user by the policies.

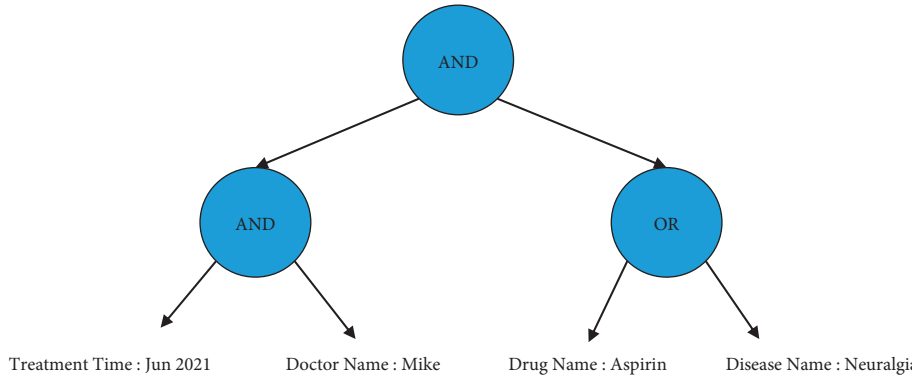

Figure 4: Boolean keyword expression.

2.6. Security Model of ASBBKS. We will give the security definition of ASBBKS. If there is no adversary who can win the following game with a non-negligible advantage in probabilistically polynomial time (PPT), then we can say that an ASBBKS scheme is secure against selectively chosen keyword attack (INDSCKA). In the following game, adversary $\mathscr{A}$ interacts with the game challenger $\mathscr{C}$ as follows.

(i) **Setup:** in this stage, challenger $\mathscr{C}$ first generates $(\mathrm{pk}, \mathrm{mk})$ through running $\mathrm{Setup}(k)$. Then, pk will be sent to adversary $\mathscr{A}$.

(ii) **Phase 1:** in this stage, adversary $\mathscr{A}$ can make a polynomial number of queries as follows:

(1) **Secret key queries:** adversary $\mathscr{A}$ adaptively queries secret keys for recursive attribute set structures to challenger $\mathscr{C}$.

(2) **Trapdoor queries:** adversary $\mathscr{A}$ adaptively queries Boolean keyword value expressions B_V to challenger $\mathscr{C}$ and gets the corresponding trapdoors.

(iii) **Challenge:** after the above queries, adversary $\mathscr{A}$ submits two distinct keyword value sets W_{V0} and W_{V1} to challenger $\mathscr{C}$. These sets cannot satisfy the Boolean keyword value expression B_V that is queried with the form of Trapdoor queries in Phase 1. Then, $\mathscr{C}$ randomly selects a bit $\beta \in \{0, 1\}$, constructs the challenge ciphertext C^* of $W_{V\beta}$, and returns it to $\mathscr{A}$.

(iv) **Phase 2:** adversary $\mathscr{A}$ continues to make secret key queries and trapdoor queries similar to phase 1. It requires both W_{V0} and W_{V1} to not satisfy the Boolean keyword value expression B_V.

(v) **Guess:** adversary $\mathscr{A}$ has to guess and output a bit $\beta' \in \{0, 1\}$. It wins the game if $\beta' = \beta$.

3. Our Construction of ASBBKS

In this section, we will provide a concrete construction of our scheme and the formal security proof. In the following construction and security proof, we assume the depth of the user's attribute set structure to be 2.

3.1. The Concrete Construction

3.1.1. Setup $(\kappa) \longrightarrow \{pk, mk\}$. Let e: $G \times G \longrightarrow G_T$ be a bilinear pairing in which G and G_T represent two cyclic groups of prime order p. Assume that g is the generator of G and H_0: $\{0,1\}^* \longrightarrow Z_p^*$ and H_1: $\{0,1\}^* \longrightarrow G$ are two collision-resistant hash functions. Next, it randomly selects $\beta_1, \beta_2, \alpha \in Z_p^*$ and calculates $h_1 = g^{\beta_1}$, $h_2 = g^{\beta_2}$, $h_3 = g^{\alpha}$. The master key and the public key $(\mathrm{pk}, \mathrm{mk})$ are set to be

$$\begin{aligned} \mathrm{pk} &= \langle G, G_T, p, g, e, h_1, h_2, h_3, H_0(\cdot), H_1(\cdot)\rangle, \\ \mathrm{mk} &= \langle \beta_1, \beta_2, \alpha\rangle. \end{aligned} \tag{1}$$

3.1.2. KeyGen $(\mathbb{A}, mk) \longrightarrow \{sk\}$. The input recursive attribute set structure is parsed as $\mathbb{A} = \{A_0, A_1, \ldots, A_n\}$. Assume that each A_i has m_i attribute members; then, there is $A_i = \{a_{i,1}, \ldots, a_{i,m_i}\}$ for $0 \le i \le n$. Firstly, it randomly selects a value $r \in Z_p^*$ for attribute set structure $\mathbb{A}$. For each subset A_i, it randomly selects n values $\{r_i \in Z_p^* \mid 1 \le i \le n\}$ and sets $r_0 = r$. Furthermore, it randomly selects a set of values $\{r_{i,j} \in Z_p^* \mid 0 \le i \le n, 1 \le j \le m_i\}$ for each attribute. Finally, it calculates $B = g^{(\alpha - r)/\beta_1}$, $B_{i,j} = g^{r_i} H_1(a_{i,j})^{r_{i,j}}$ and $B'_{i,j} = g^{r_{i,j}}$ for $0 \le i \le n$, $1 \le j \le m_i$ and $D_i = g^{(r + r_i)/\beta_2}$ for $1 \le i \le n$. The secret key for $\mathbb{A}$ is set to be

$$\begin{aligned} \mathrm{sk} = \langle \mathbb{A}, B, \{(B_{i,j}, B'_{i,j}) \mid 0 \le i \le n, 1 \le j \le m_i\}, \\ \{D_i \mid 1 \le i \le n\}\rangle. \end{aligned} \tag{2}$$

At the translating node, each element D_i in a secret key supporting r_i of set A_i at depth 2 translates to r of set A_0 at depth 1. Elements D_i and $D_{i'}$ can be combined as $D_i/D_{i'}$ to translate $r_{i'}$ to r_i at the translating nodes.

3.1.3. Encrypt $(pk, W_V, \mathcal{T}) \longrightarrow \{C\}$. $W_V = (w_{\rho(1)}, w_{\rho(2)}, \ldots, w_{\rho(m)})$ is a set of keyword values and $W_N = (\rho(1), \rho(2), \ldots, \rho(m))$ is the set of the keyword names. This algorithm randomly selects $m+1$ values $s_0, s_1, \ldots, s_m \in Z_p^*$ and computes $C_1 = h_1^{s_0}$, $C_2 = h_2^{s_0}$, $C_{i,1} = g^{s_i}$, $C_{i,2} = h_3^{(s_0 + s_i)} \cdot h_2^{H_0(w_{\rho(i)})s_i}$ for

$1 \le i \le m$. After that, it computes secret shares of s_0 through implementing $\{q_v(0) \mid v \in \mathrm{lvs}(\mathcal{T})\} \longleftarrow \mathrm{Share}(\mathcal{T}, s_0)$. Further, for each $v \in \mathrm{lvs}(\mathcal{T})$, it computes $C_v = g^{q_v(0)}$ and $C_v' = H_1(\mathrm{att}(v))^{q_v(0)}$. We assume that the set of translating nodes in $\mathcal{T}$ is represented as $\mathrm{trans}(\mathcal{T})$. Then, it calculates $\widetilde{C}_x = h_2^{q_x(0)}$ for each $x \in \mathrm{trans}(\mathcal{T})$. Finally, the ciphertext is set to be

$$C = \langle \mathcal{T}, W_N, C_1, C_2, \{(C_{i,1}, C_{i,2}) \mid 1 \le i \le m\}, \{(C_v, C_v') \mid v \in \mathrm{lvs}(\mathcal{T})\}, \{\widetilde{C}_x \mid x \in \mathrm{trans}(\mathcal{T})\}\rangle. \tag{3}$$

In user keys, elements $(\widetilde{C}_x)'s$ and $(D_i)'s$ support translation between sets at a translating node x. We will describe it later in the Test algorithm.

3.1.4. Trapdoor $(sk, \mathscr{B}_V) \longrightarrow \{T\}$. $\mathscr{B}_V$ represents the Boolean keyword value expression which is an access tree. $\mathscr{B}_N$ represents the Boolean keyword name expression which is an access tree with the same structure as $\mathscr{B}_V$. Taking sk and $\mathscr{B}_V$ as inputs, it randomly selects a value $t \in Z_p^*$ and calculates the secret shares of t by implementing $\{\widehat{q}_v(0) \mid v \in \mathrm{lvs}(\mathscr{B}_N)\} \longleftarrow \mathrm{Share}(\mathscr{B}_N, t)$. Then, this algorithm calculates $T_{\rho(v),1} = (h_3 h_2^{(H_0 \overline{w}_{\rho(v)})})^{\widehat{q}_v(0)}$ and $T_{\rho(v),2} = g^{\widehat{q}_v(0)}$. Parsing sk as $\langle \mathbb{A}, B, \{(B_{i,j}, B_{i,j}') \mid 0 \le i \le n, 1 \le j \le m_i\}, \{D_i \mid 1 \le i \le n\}\rangle$, it further computes $\overline{B} = B^t, \overline{B}_{i,j} = B_{i,j}^t, \overline{B'}_{i,j} = B_{i,j}'^t$ for $0 \le i \le n, 1 \le j \le m_i$ and $\overline{D}_i = D_i^t$ for $1 \le i \le n$. Finally, the trapdoor for $\mathscr{B}_V$ is

$$T = \langle \mathbb{A}, \mathscr{B}_N, \{(T_{\rho(v),1}, T_{\rho(v),2}) \mid v \in \mathrm{lvs}(\mathscr{B}_N)\}, \overline{B}, \{(\overline{B}_{i,j}, \overline{B'}_{i,j}) \mid 0 \le i \le n, 1 \le j \le m_i\}, \{\overline{D}_i \mid 1 \le i \le n\}\rangle. \tag{4}$$

3.1.5. Test $(C, T) \longrightarrow \{0, 1\}$. Cloud server takes input ciphertext $C = \langle \mathcal{T}, W_N, C_1, C_2, \{(C_{i,1}, C_{i,2}) \mid 1 \le i \le m\}, \{(C_v, C_v') \mid v \in \mathbb{Y}\}, \{\widetilde{C}_x \mid x \in \mathrm{trans}(\mathcal{T})\}\rangle$ and the trapdoor $\underline{T} = \langle \mathbb{A}, \mathscr{B}_N, \{(T_{\rho(v),1}, T_{\rho(v),2}) \mid v \in \mathrm{lvs}(\mathscr{B}_N)\}, \overline{B}, \{(\overline{B}_{i,j}, \overline{B'}_{i,j}) \mid 0 \le i \le n, 1 \le j \le m_i\}, \{\overline{D}_i \mid 1 \le i \le n\}\rangle$. The complete Test algorithm consists of the following 3 steps.

Step 1. In this step, for the given access tree $\mathcal{T}$ and key structure $\mathbb{A}$, it returns a set S_τ. The elements of this set are some labels for each node τ within $\mathcal{T}$. Each label t in S_τ represents a set A_t and each set satisfies the subtree $\mathcal{T}_\tau$. There is $\mathcal{T}_R = \mathcal{T}$ for the root node R and the related set is S_R. If $\mathbb{A}$ does not satisfy $\mathcal{T}$, the return of this algorithm is "0." Otherwise, for node τ, it picks one label from the set S_τ, denoted as i. Then, it executes a recursive function $\mathrm{DecryptNode}(C, T, \tau, i)$, which will return F_τ as the result. According to the type of node τ, the function $\mathrm{DecryptNode}(C, T, \tau, i)$ will be computed in two different ways.

(1) When τ is a leaf node: if $\mathrm{att}(\tau) \notin A_i$, it returns "$\perp$". Otherwise, we have

$$\mathrm{DecryptNode}(C, T, \tau, i) = \frac{e(C_\tau, \overline{B}_{i,j})}{e(C_\tau', \overline{B'}_{i,j})} = e(g, g)^{t \cdot r_i \cdot q_\tau(0)}. \tag{5}$$

(2) When τ is not a leaf node:

(i) Firstly, it calculates a set E_τ, which is composed of k_τ child nodes of τ. In E_τ, each node z must satisfy one of the following two cases: (1) label $i \in S_z$ and (2) z is a translating node and there exists a label i' that satisfies $i' \in S_z$ and $i' \ne i$. If such a set does not exist, it returns "$\perp$".

(ii) Execute $\mathrm{DecryptNode}(C, T, z, i)$ for each node $z \in E_\tau$ which satisfies label $i \in S_z$ and then return F_z as the result.

(iii) Execute $\mathrm{DecryptNode}(C, T, z, i')$ for each translating node $z \in E_\tau$ which satisfies $i' \in S_z$ and $i' \ne i$ and return F_z' as the result.
If $i = 0$, elements $\overline{D}_{i'}$ and $\widetilde{C}_z$ can be used to translate F_z' to F_z.

$$F_z = \frac{e(\overline{D}_{i'}, \widetilde{C}_z)}{F_z'} = \frac{e\left(g^{t\left(r + r_{i'}\right)/\beta_2}, g^{\beta_2 \cdot q_z(0)}\right)}{e(g, g)^{t \cdot r_{i'} \cdot q_z(0)}} = e(g, g)^{t \cdot r \cdot q_z(0)}. \tag{6}$$

If $i \ne 0$, elements $\overline{D}_i$ and $\overline{D}_{i'}$ together with $\widetilde{C}_z$ can be used to translate F_z' to F_z.

$$F_z = e\left(\frac{\overline{D}_i}{\overline{D}_{i'}}, \widetilde{C}_z\right) \cdot F_z' = e\left(g^{\frac{t(r_i - r_{i'})}{\beta_2}}, g^{\beta_2 \cdot q_z(0)}\right) e(g, g)^{t \cdot r_{i'} \cdot q_z(0)} = e(g, g)^{t \cdot r_i \cdot q_z(0)}. \tag{7}$$

(iv) After computing F_z for each node z in E_τ, it computes F_τ as follows:

$$F_\tau = \prod_{z \in E_\tau} F_z^{\Delta_{i,U_z}(0)} = \begin{cases} e(g, g)^{t \cdot r \cdot q_\tau(0)}, & i = 0 \\ e(g, g)^{t \cdot r_i \cdot q_\tau(0)}, & i \ne 0 \end{cases}, \tag{8}$$

where $\Delta_{i,S}(x) = \prod_{j \in S, j \ne i} (x - j)/(i - j)$, $v = \mathrm{label}(z)$, and $U_z = \{\mathrm{label}(z) : z \in E_\tau\}$.

After the above steps, it further computes the function $\mathrm{DecryptNode}(C, T, R, i)$ for root node R and returns F_R as the result as follows.

$$F_R = \begin{cases} e(g,g)^{t\cdot r\cdot s_0}, & i = 0, \\ e(g,g)^{t\cdot r_i\cdot s_0}, & i \neq 0, \end{cases} \tag{9}$$

In the end, it computes a value F. When $i = 0$, it assumes $F = F_R$. When $i \neq 0$, it computes F as follows:

$$F = \frac{e(\overline{D}_i, C_3)}{F_R} = \frac{e(g,g)^{t\cdot(r+r_i)\cdot s_0}}{e(g,g)^{t\cdot r_i\cdot s_0}} = e(g,g)^{trs_0}. \tag{10}$$

Step 2. Firstly, it computes a value L with the following formula:

$$L = e(\overline{B}, C_1) = e\left(g^{t(\alpha-r)/\beta_1}, g^{\beta_1 s_0}\right) = e(g,g)^{t(\alpha-r)s_0}. \tag{11}$$

Then, it computes $F \cdot L$ with the following formula:

$$F \cdot L = e(g,g)^{\mathrm{trs}_0} e(g,g)^{t(\alpha-r)s_0} = e(g,g)^{t\alpha s_0}. \tag{12}$$

Step 3. It selects a minimum subset W_N' from the set of keyword names W_N, which satisfies the Boolean keyword name expression $\mathscr{B}_N$. If W_N' does not exist, it returns "0." If node $\widehat{x}$ is a leaf node of access tree $\mathscr{B}_N$ and is related to the search token T, then the keyword name associated with this node is denoted by $\rho(\widehat{x})$. Further, for each keyword name $\rho(\widehat{x}) \in W_N'$, it computes

$$\begin{aligned} E_{\widehat{x}} &= \frac{e\left(C_{\rho(\widehat{x}),2}, T_{\rho(\widehat{x}),2}\right)}{e\left(T_{\rho(\widehat{x}),1}, C_{\rho(\widehat{x}),1}\right)} \\ &= \frac{e\left(h_3^{s_0+s_{\rho(\widehat{x})}} h_2^{H_0\left(w_{\rho(\widehat{x})}\right)s_{\rho(\widehat{x})}}, g^{\widehat{q}_{\widehat{x}}(0)}\right)}{e\left(\left(h_3 h_2^{H_0\left(\overline{w}_{\rho(\widehat{x})}\right)}\right)^{\widehat{q}_{\widehat{x}}(0)}, g^{s_{\rho(\widehat{x})}}\right)}. \end{aligned} \tag{13}$$

If $w_{\rho(\widehat{x})} = \overline{w}_{\rho(\widehat{x})}$, then there is $E_{\widehat{x}} = e(g,g)^{\alpha s_0 \widehat{q}_{\widehat{x}}(0)}$.

When node $\widehat{x}$ is not a leaf node, for all child nodes $\widehat{z}$ of $\widehat{x}$, assume that $S_{\widehat{x}}$ represents an arbitrary set with size $k_{\widehat{x}}$ consisting of children nodes $\widehat{z}$ and $E_{\widehat{z}} \neq \perp$. If $S_{\widehat{x}}$ does not exist, then $E_{\widehat{z}} \neq \perp$; otherwise, it utilizes the polynomial interpolation to calculate $E_{\widehat{x}}$ to get

$$E_{\widehat{x}} = e(g,g)^{\alpha s_0 \widehat{q}_{\widehat{x}}(0)}. \tag{14}$$

At last, for the root node of $\mathscr{B}_N$, it computes $E_{\widehat{R}}$ and checks whether the following equation holds:

$$E_{\widehat{R}} = F \cdot L. \tag{15}$$

If the equation above does not hold, then the algorithm would keep finding another subset of keyword names from the set of keyword names W_N which satisfies $\mathscr{B}_N$ and repeat the checking as above. If there exists no such keyword name subset such that the above equation holds, it returns "$\perp$".

We depict the whole processing steps of our scheme in Figure 5.

Remark 1. For the user's attribute set structure with depth d, for each level i, a value β_i needs to be selected. β_i supports the translations between sets at level i or between a set at level i and its outer set at level $i-1$. Translations across multiple levels will use corresponding translating values and different βs.

3.2. Security Proof. In this section, we will provide the formal proof of Theorem 1 to prove that our scheme is secure.

Theorem 1. *The above scheme is selectively secure against chosen keyword attack in the generic bilinear group model.*

Proof. In our security model, the adversary $\mathscr{A}$ assumes to distinguish $g^{\alpha(s_0+s_i)} g^{\beta_2 H_0(w^0_{\rho(i)})s_i}$ from g^θ and $g^{\alpha(s_0+s_i)} g^{\beta_2 H_0(w^1_{\rho(i)})s_i}$ from g^θ, where θ is randomly selected from Z_p^*, W_0 and W_1 are two different keyword sets, $w^0_{\rho(i)} \in W_0$, and $w^1_{\rho(i)} \in W_1$. Since $\mathscr{A}$ has the same probability to distinguish both of them, it is easy to distinguish $g^{\beta_2 H_0(w^0_{\rho(i)})s_i}$ from g^θ. That is, this game can be transformed to $\mathscr{A}$ that has the advantage $\epsilon/2$ to distinguish $g^{\alpha(s_0+s_i)}$ from g^θ.

The challenger $\mathscr{C}$ can be constructed in this game as follows.

(i) **Setup**: the challenger $\mathscr{C}$ selects parameters α, β_1, $\beta_2 \in Z_p^*$ and sends public parameter $\mathrm{pk} = \langle G, G_T, p, g, e, g^{\beta_1}, g^{\beta_2}, g^\alpha, H_0\rangle$ to $\mathscr{A}$. After that, $\mathscr{A}$ selects a challenge access tree $\mathscr{T}^*$ and returns it to $\mathscr{C}$.

(ii) Hash_1-**Queries**: challenger $\mathscr{C}$ would maintain an H-list, which is empty at first. Given an input attribute $a_{i,j}$, if $a_{i,j}$ has not been queried, it randomly selects $t_{i,j} \in Z_p^*$ and then returns $g^{t_{i,j}}$ to $\mathscr{A}$ and the tuple $(a_{i,j}, t_{i,j})$ will be added to H-list. For those attributes $a_{i,j}$ that have been queried, $\mathscr{C}$ will directly return $g^{t_{i,j}}$ to $\mathscr{A}$. $\mathscr{A}$ can query this random oracle for polynomially many times. □

Phase 1. Adversary $\mathscr{A}$ can make the secret key and trapdoor queries for polynomially many times.

(i) **Secret key queries:** firstly, for an attribute set structure $\mathbb{A} = \{A_0, A_1, \ldots, A_n\}$ where $A_i = \{a_{i,1}, \ldots, a_{i,m_i}\}$ for $0 \le i \le n$, $\mathscr{C}$ will randomly select a value $r \in Z_p^*$ for $\mathbb{A}$ and compute $B = g^{(\alpha-r)/\beta_1}$. For each subset A_i, it will randomly select $r_i \in Z_p^*$ and a value $r_{i,j} \in Z_p^*$ for $0 \le i \le n$, $1 \le j \le m_i$. Next, use these parameters to compute corresponding $B_{i,j} = g^{r_i} g^{t_{i,j} r_{i,j}}$, $B_{i,j}' = g^{r_{i,j}}$, and $D_i = g^{(r+r_i)/\beta_2}$. Finally, $\mathscr{C}$ constructs the secret key $\mathrm{sk} = \langle \mathbb{A}, B, \{(B_{i,j}, B_{i,j}') \mid 0 \le i \le n, 1 \le j \le m_i\}, \{D_i \mid 1 \le i \le n\}\rangle$ with the above parameters and returns it to $\mathscr{A}$.

(ii) **Trapdoor queries**: if $\mathscr{A}$ has queried secret key sk, then $\mathscr{C}$ directly runs the Trapdoor algorithm to get

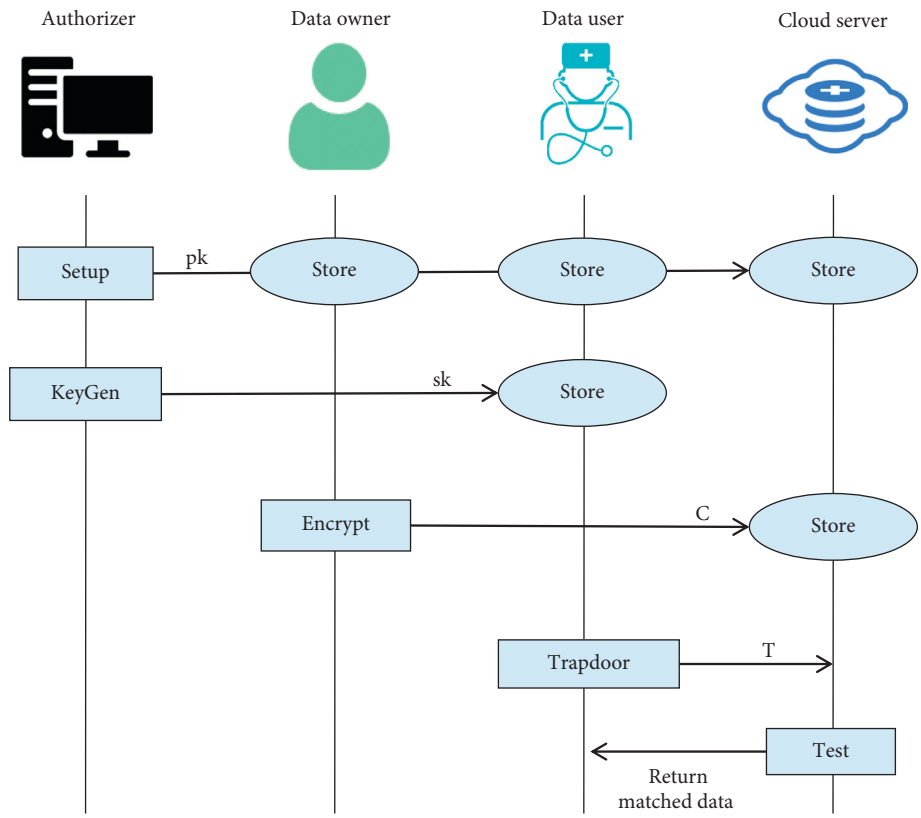

Figure 5: Workflow of our scheme.

the trapdoor T. Challenger $\mathscr{C}$ randomly selects $t \in Z_p^*$ and computes the secret shares of t by running $\{\widehat{q}_v(0) \mid v \in \mathrm{lvs}(\mathscr{B}_N)\} \longleftarrow \mathrm{Share}(\mathscr{B}_N, t)$ and computes corresponding $T_{\rho(v),1} = (g^{\alpha} g^{\beta_2 H_0(\overline{w}_{\rho(v)})})^{\widehat{q}_v(0)}$, $T_{\rho(v),2} = g^{\widehat{q}_v(0)}$, $\overline{B} = B^t, \overline{B}_{i,j} = B_{i,j}^t$, $\overline{B'}_{i,j} = B'^t_{i,j}$, and $\overline{D}_i = D_i^t$. Then the trapdoor for Boolean keyword value expression $\mathscr{B}_V$ is $T = \langle \mathbb{A}, \mathscr{B}_N, \{(T_{\rho(v),1}, T_{\rho(v),2}) \mid v \in \mathrm{lvs}(\mathscr{B}_N)\}, \overline{B}, \{(\overline{B}_{i,j}, \overline{B'}_{i,j}) \mid 0 \le i \le n, 1 \le j \le m_i\}, \{\overline{D}_i \mid 1 \le i \le n\}\rangle$, and the challenger C returns it to $\mathscr{A}$.

Challenge: $\mathscr{A}$ submits two distinct keyword value sets $W_{V0} = (w^0_{\rho(1)'}, w^0_{\rho(2)'}, \ldots, w^0_{\rho(m)})$ and $W_{V1} = (w^1_{\rho(1)'}, w^1_{\rho(2)'} \cdots, w^1_{\rho(m)})$ to $\mathscr{C}$, and these sets are of equal size. Then, let $W_N = (\rho(1), \rho(2), \ldots, \rho(m))$ represent the set of keyword name. For these two sets, they cannot satisfy the Boolean keyword value expression B_V. Then, $\mathscr{C}$ randomly selects $s_0, s_i \in Z_p^*$, for $1 \le i \le m$ and computes secret shares of s_0 by executing $Share(\mathscr{T}^*, s_0) \longrightarrow \{q_v(0) \mid v \in \mathrm{lvs}(\mathscr{T}^*)\}$.

Then, C randomly selects a bit $\beta \in \{0, 1\}$ and constructs the challenge ci-phertext $C^* = \langle \mathscr{T}^*, W_N, C_1, C_2, \{(C_{i,1}, C_{i,2}) \mid 1 \le i \le m\}, \{(C_v, C_v') \mid v \in \mathrm{lvs}(\mathscr{T})\}, \{\widetilde{C}_x \mid x \in \mathrm{trans}(\mathscr{T}^*)\}\rangle$, and then C returns it to $\mathscr{A}$.

(i) If $\beta = 0$, it randomly selects $\theta \in Z_p^*$ and outputs $C_1 = g^{\beta_1 s_0}, C_2 = g^{\beta_2 s_0}$, for $1 \le i \le m$, $C_{i,1} = g^{s_i}, C_{i,2} = g^{\theta}$, $\{(C_v = g^{q_v(0)}, C_v' = g^{t_{i,j} q_v(0)} \mid v \in \mathrm{lvs}(\mathscr{T}^*), \mathrm{att}(v) = a_{i,j})\}$, $\{\widetilde{C}_x = g^{\beta_2 q_x(0)} \mid x \in \mathrm{trans}(\mathscr{T}^*)\}$

(ii) If $\beta = 1$, it outputs $C_1 = g^{\beta_1 s_0}$, $C_2 = g^{\beta_2 s_0}$, for $1 \le i \le m, C_{i,1} = g^{s_i}, C_{i,2} = g^{\alpha(s_0 + s_i)}$, $\{(C_v = g^{q_v(0)}, C_v' = g^{t_{i,j} q_v(0)} \mid v \in \mathrm{lvs}(\mathscr{T}^*), \mathrm{att}(v) = a_{i,j})\}$, $\{\widetilde{C}_x = g^{\beta_2 q_x(0)} \mid x \in \mathrm{trans}(\mathscr{T}^*)\}$.

Phase 2. After receiving the challenge ciphertext C^*, $\mathscr{A}$ can perform secret key queries and trapdoor queries mentioned in phase 1. It requires both W_{V0} and W_{V1} to not satisfy the Boolean keyword value expression B_V.

Guess: $\mathscr{A}$ outputs its guess, which requires $\mathscr{A}$ to distinguish $g^{(s_0 + s_i)}$ from g^{θ} to win this game.

Analysis: if $\mathscr{A}$ can construct $e(g^{\eta}, g^{\alpha(s_0 + s_i)})$ for some g^{η}, $\mathscr{A}$ has the ability to distinguish $g^{\alpha(s_0 + s_i)}$ from a random element g^{θ}. Thus, we continue to analyze that adversary $\mathscr{A}$ constructs $e(g, g)^{\eta\alpha(s_1 + s_2)}$ for some g^{η} with negligible advantage. That means $\mathscr{A}$ cannot win our game with a non-negligible advantage.

From the above phases, it can be easily found that s_i and α have appeared, so $\mathscr{A}$ only needs to construct αs_0. With

terms $g^{\beta_1 s_0}$ and $g^{(\alpha-r)/\beta_1}$, $\mathscr{A}$ can construct $e(g,g)^{\alpha s_0} e(g,g)^{-rs_0}$ and then $\mathscr{A}$ needs to conceal $e(g,g)^{rs_0}$. With terms $g^{\beta_2 s_0}$ and $g^{(r+r_i)/\beta_2}$, $\mathscr{A}$ can construct $e(g,g)^{(\alpha s_0 + r_i s_0)}$ and $\mathscr{A}$ needs to conceal $e(g,g)^{r_i s_0}$. Then, $g^{r_i s_0}$ should be constructed which will use terms $g^{r_i + a_{ij} ri}$, $g^{r_{ij}}$, $g^{q_v(0)}$, and $g^{a_{i,j} q_v(0)}$ because $q_v(0)$ is the secret share of s_0. However, $g^{r_i s_0}$ cannot be constructed since there are not enough attribute values $g^{q_v(0)}$ for $\mathscr{A}$ to satisfy the access tree $\mathscr{T}$.

4. Performance Analysis

In the existing ABKS schemes, no scheme can support both Boolean keyword search and recursive set structure. At first, we compare the existing schemes [1, 7, 28] in Table 1 with our scheme in the aspect of their functionalities. We assume four parts to compare, search method, access structure, attribute set structure, and translating nodes. From Table 1, we can conclude that our scheme is the first one that supports compound attributes, flexible access policies' specifying, and Boolean keyword search simultaneously.

4.1. Theoretical Analysis. In our scheme, three operations are the most time consuming which are, respectively, the bilinear pairing, the modular exponentiation, and the hash function H_1. Since H_1 can be precomputed, we just consider the former two operations in the following analysis.

In theoretical analysis, we focus on the computation complexity and storage overhead of each step. Firstly, we assume a user's 2-level recursive attribute set to be $\mathbb{A} = \{A_0, A_1, \ldots, A_n\}$ where $A_i = \{a_{i,1}, a_{i,2}, \ldots, a_{i,m_i}\}$. Let $M = m_0 + m_1 + \cdots + m_n$. Then, let the leaf nodes of an access tree $\mathscr{T}$ be $|\mathrm{lvs}(\mathscr{T})|$ and translating nodes be $|\mathrm{trans}(\mathscr{T})|$. For the keyword set used in Encrypt, we denote the number of keywords in this set to be N and the leaf nodes of a Boolean keyword value expression B_V to be $|\mathrm{lvs}(B_V)|$. We organize the theoretical computation complexity and storage overhead of the existing schemes [1, 7, 28] and our scheme in Table 2.

KeyGen. For the KeyGen algorithm in our scheme, its computation complexity is $2M + 2n + 1$ exponentiations in G. The storage overhead is $2M + n + 1$ group elements in G. In [28], the KeyGen algorithm takes $2M + 1$ exponentiations in G. The storage overhead is $2M$ group elements in G. In [7], the computation complexity of its KeyGen algorithm is $1 + 2n + 2M$ exponentiations in G. The storage overhead is $2M + n + 1$ group elements in G. In [1], the KeyGen algorithm takes $2 + 2n$ exponentiations in G. The storage takes $2M + 1$ group elements in G.

Encrypt. For the Encrypt algorithm in our scheme, its computation complexity is $|\mathrm{trans}(\mathscr{T})| + 2|\mathrm{lvs}(\mathscr{T})| + 3N + 2$ exponentiations in G. The storage overhead is $|\mathrm{trans}(\mathscr{T})| + 2|\mathrm{lvs}(\mathscr{T})| + 2N + 2$ group elements in G. In [28], the Encrypt algorithm takes $2|\mathrm{lvs}(\mathscr{T})| + 4$ exponentiations in G. The storage overhead is $2|\mathrm{lvs}(\mathscr{T})| + 3$ group elements in G.

Table 1: Comparison of functionalities between SE schemes.

	Boolean keyword search	Access structure	Compound attributes	Translating nodes
[1]	√	LSSS	×	×
[28]	×	Access tree	×	×
[7]	×	Access tree	√	√
Our ASBBKS	√	Access tree	√	√

In [28], the computation complexity of its Encrypt algorithm is $|\mathrm{trans}(\mathscr{T})| + 2|\mathrm{lvs}(\mathscr{T})| + 5$ exponentiations in G. The storage overhead is $|\mathrm{trans}(\mathscr{T})| + 2|\mathrm{lvs}(\mathscr{T})| + 4$ group elements in G. In [28], the Encrypt algorithm takes $1 + 2|\mathrm{lvs}(\mathscr{T})| + 3N$ exponentiations in G. The storage takes $1 + 2|\mathrm{lvs}(\mathscr{T})| + 2N$ group elements in G.

Trapdoor. For the Trapdoor algorithm in our scheme, its computation complexity is $2M + n + 3|\mathrm{lvs}(B_N)| + 1$ exponentiations in G. The storage overhead is $2M + n + 1 + 2|lvs(B_N)|$ group elements in G. In [1], the Trapdoor algorithm takes $2M + 4$ exponentiations in G. The storage overhead is $2M + 3$ group elements in G. In [28], the computation complexity of its Trapdoor algorithm is $4 + n + 2M$ exponentiations in G. The storage overhead is $3 + n + 2M$ group elements in G. In [7], the Trapdoor algorithm takes $1 + 2M + 3|\mathrm{lvs}(B_N)|$ exponentiations in G. The storage takes $1 + 2M + 2|\mathrm{lvs}(B_N)|$ group elements in G.

Test. For the Test algorithm in our scheme, we first assume $\mathbb{A}$ has l leaf nodes satisfying $\mathscr{T}$ and k_i translating nodes on the path from ith leaf node used to the root node where $1 \le i \le l$, and let $k = k_1 + k_2 + \cdots + k_l$. In addition, we denote the number of all used nodes in $\mathscr{T}$ as t, and the computation complexity is $2l + k + 1 + 2|\mathrm{lvs}(B_N)|$ pairings and $t - 1 + |\mathrm{lvs}(B_N)|$ exponentiations in G_T. The storage overhead depends on the number of the matched ciphertexts, which we denote as NC. In [28], the Test algorithm takes $2l + 3$ pairings and $t - 1$ exponentiations in G_T and the storage overhead we assume is NC. In [7] the computation complexity of its Test algorithm is $3 + k + 2l$ pairings and $t - 1$ exponentiations in G_T, and the storage overhead we assume is NC. In [1], the *Search* algorithm takes $1 + 2l + 2|\mathrm{lvs}(B_N)|$ parings and $\mathrm{lvs}|\mathscr{T}| + |\mathrm{lvs}(B_N)|$ exponentiations in G_T and the storage overhead we assume is NC.

Remark 2. From the analysis, the difference in computation cost between our scheme and [28] depends linearly on the number of keywords, the number of inner sets in $\mathbb{A}$, and the number of translating nodes in $\mathscr{T}$. When there is only one set in $\mathbb{A}$, one keyword in the Encrypt algorithm and Trapdoor algorithm, and no translating node in access tree $\mathscr{T}$, the computation cost of our scheme is the same as that of the scheme in [28]. The difference in computation cost between our scheme and [7] depends

Table 2: Theoretical efficiency analysis and comparison between schemes.

		[31]	[28]	[11]	Our scheme
Computation costs	KeyGen	$(2M+1)E$	$(1+2n+2M)E$	$(2+2n)E$	$(1+2n+2M)E$
	Encrypt	$(2lvs\|T\|+4)E$	$(\|\text{trans}(T)\|+2lvs\|T\|+5)E$	$(1+2lvs\|T\|+3N)E$	$(\|\text{trans}(T)\|+2lvs\|T\|+3N+2)E$
	Trapdoor	$(2M+4)E$	$(4+n+2M)E$	$(1+2M+3\|lvs(B_N)\|)E$	$(1+3\|lvs(B_N)\|+n+2M)E$
	Test	$(2l+3)e+(t-1)E_T$	$(2l+k+3)e+(t-1)E_T$	$(1+2l+2\|lvs(B_N)\|)e+(lvs\|T\|+\|lvs(B_N)\|)E_T$	$(2l+k+1+2\|lvs(B_N)\|)e+(t-1+\|lvs(B_N)\|)E_T$
Communication costs	KeyGen	$(2M)G$	$(1+n+2M)G$	$(1+2M)G$	$(1+n+2M)G$
	Encrypt	$(2\|lvs\|\|T\|+3)G$	$(2lvs\|T\|+\|\text{trans}(T)\|+4)G$	$(1+2lvs\|T\|+2N)G$	$(2lvs\|T\|+\|\text{trans}(T)\|+2N+2)G$
	Trapdoor	$(2M+3)G$	$(3+n+2M)G$	$(1+2M+2\|lvs(B_N)\|)G$	$(1+n+2M+2\|lvs(B_N)\|)G$
	Test	NC	NC	NC	NC

e: evaluation of a bilinear pairing; E: evaluation of a modular exponentiation in G; E_T: evaluation of a modular exponentiation in G_T; M: number of attributes in $\mathbb{A}$; n: number of sets in $\mathbb{A}$; l: number of attributes used in $\mathbb{A}$ required to satisfy T; t: number of all used nodes in T; $|lvs(T)|$: number of leaf nodes used in T; $|lvs(B_N)|$: number of keywords used in B_N; $|\text{trans}(T)|$: number of translating nodes used in T; k: number of all translating nodes on the path from each leaf node used to the root.

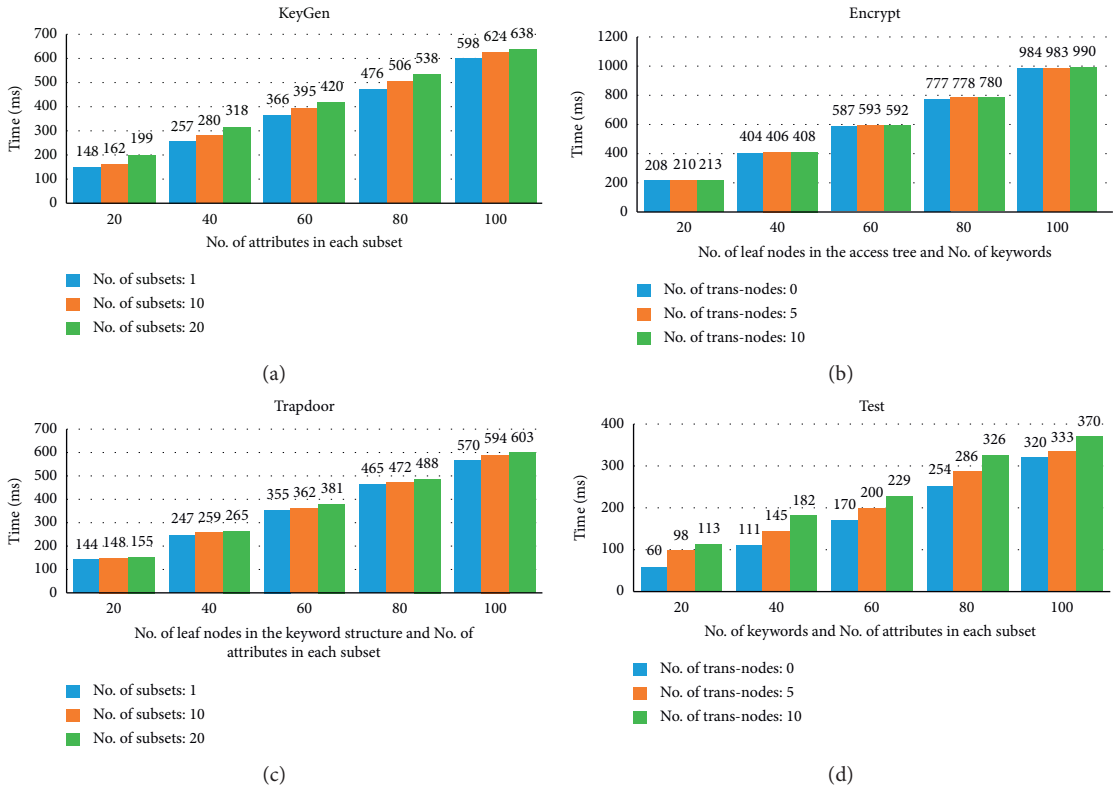

Figure 6: Operation time of the algorithms in ASBBKS. (a) KeyGen. (b) Encrypt. (c) Trapdoor. (d) Test.

linearly on the number of keywords. When there is only one keyword in the Encrypt algorithm and Trapdoor algorithm, the computation cost of our scheme is the same as that of the scheme in [7]. The difference in computation cost between our scheme and [1] depends linearly on the number of inner sets in $\mathbb{A}$ and the number of translating nodes in $\mathcal{T}$. When there is one set in $\mathbb{A}$ and no translating node in access tree $\mathcal{T}$, the computation cost of our scheme is the same as that of the scheme in [1]. In summary, our ASBBKS scheme supports compound attributes and Boolean queries by adding some computation operations, which is more suitable for practical application environments.

4.2. Experiments. In the following, we have made a series of comprehensive experiments to simulate the execution of our scheme on a personal computer (the CPU is i7-8700U 3.2 GHz with a 24 GB memory and the operating system is Ubuntu 18.04 LTS) and the PBC library. We evaluate our scheme with different parameters and record the execution time of each step in our ASBBKS scheme. In these experiments, we assume that the user's recursive attribute set $\mathcal{A}$ has two levels. We show our experiment results of each step in Figure 6.

KeyGen. Figure 6(a) shows the time cost of the KeyGen phase for our ASBBKS scheme. We consider two parameters that may affect the execution time of this step: the number of subsets in a user's set structure and the number of attributes in each subset. We assume the number of subsets to be 1, 10, and 20 and the number of attributes in each subset to be 20, 40, 60, 80, and 100, respectively. As we can see, the time cost of KeyGen is increased linearly with the above two parameters. It only takes 638 ms when there are 20 subsets and 100 attributes in each subset. The number of attributes in each subset influences the execution time most, while the number of subsets has a smaller impact.

Encrypt. Figure 6(b) shows the time cost of the Encrypt phase. We consider three parameters that may affect the execution time of this step: the number of translating nodes and leaf nodes in an access tree and the number of keywords. We assume that there are 20, 40, 60, 80, and 100 leaf nodes and keywords and there are 0, 5, and 10 translating nodes, respectively. As we can see, the time cost of Encrypt is increased linearly with the above parameters. It only takes 990 ms when there are 10 translating nodes and 100 leaf nodes in an access tree and 100 keywords in a document. In these parameters, the main influencing factors are the number of leaf nodes and the number of keywords, while the number of translating nodes only affects the execution time for several milliseconds.

Trapdoor. Figure 6(c) shows the time cost of the Trapdoor phase. We consider three parameters that may affect the execution time of this step: the number of subsets in a user's set structure, the number of attributes in each subset, and the number of leaf nodes in a keyword structure. We assume the number of attributes and leaf nodes to be 20, 40, 60, 80, and 100 and the number of subsets to be 1, 10, and

20, respectively. As we can see, the execution time of the Trapdoor phase is increased linearly with the above parameters. It only takes 603 ms when there are 20 subsets with 100 attributes in each subset and 100 leaf nodes for the keyword structure. In these parameters, the main influencing factors are the number of attributes and the number of leaf nodes, while the number of subsets has a smaller impact.

Test. Figure 6(d) shows the time cost of the Test phase. We consider three parameters that may affect the execution time of this step: the number of keywords, the number of attributes in each subset, and the number of translating nodes. We assume that there are 20, 40, 60, 80, and 100 attributes and keywords. We also assume that there are 0, 5, and 10 translating nodes. Further, we assume that the number of subsets in a user's set structure is fixed to 10. As we can see, the execution time of the Test phase is increased linearly with the above parameters. It only takes 370 ms when there are 100 attributes in each subset, 10 translating nodes in an access tree, and 100 keywords in the document. In these parameters, the main influencing factors are the number of attributes and the number of keywords, while the number of translating nodes has a certain impact on it.

From the above experiments for each algorithm in our scheme, we can conclude that it is an efficient and practical scheme for use in a PHR application.

5. Conclusion

In this paper, we present an attribute set-based Boolean keyword search over encrypted PHR. In our scheme, each data user's attributes are organized as recursive set structure, which enables more flexibility in user attribute organization and more efficiency in specifying policies than the existing ABKS schemes. Meanwhile, all authorized users can perform Boolean keyword search over the encrypted PHR. We have proved the security of our scheme formally. The experimental results show that it is feasible and practical for PHR systems.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study was supported by the Key Areas R&D Program of Science and Technology Program of Guangzhou (202 103 010 005) and Shenzhen Science and Technology Program (JCYJ20210324100813034).

References

[1] K. He, J. Guo, J. Weng, J. Weng, J. K. Liu, and X. Yi, "Attribute-based hybrid boolean keyword search over outsourced encrypted data," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1207–1217, 2018.

[2] K. Kaitai, W. Liang, and W. Susilo, "Searchable attribute-based mechanism with efficient data sharing for secure cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 9, pp. 1981–1992, 2015.

[3] 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 *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds., Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 353–373, 2013.

[4] Y. Miao, J. Ma, X. Liu, X. Li, Q. Jiang, and J. Zhang, "Attribute-based keyword search over hierarchical data in cloud computing," *IEEE Transactions on Services Computing*, vol. 13, no. 6, pp. 985–998, 2020.

[5] 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, no. 3, pp. 1080–1094, 2021.

[6] L. Xu, W. Li, F. Zhang, R. Cheng, and S. Tang, "Authorized keyword searches on public key encrypted data with time controlled keyword privacy," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2096–2109, 2020.

[7] L. Xu, X. Chen, F. Zhang et al., "ASBKS: towards attribute set based keyword search over encrypted personal health records," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2941–2952, 2021.

[8] H. Yin, J. Zhang, Y. Xiong et al., "CP-ABSE: a ciphertext-policy attribute-based searchable encryption scheme," *IEEE Access*, vol. 7, pp. 5682–5694, 2019.

[9] U. S. Varri, S. Kumar Pasupuleti, and K. V. Kadambari, "Key-escrow free attribute-based multi-keyword search with dynamic policy update in cloud computing," in *Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pp. 450–458, IEEE, Melbourne, VIC, Australia, 11 May 2020.

[10] H. Wang, X. Dong, and Z. Cao, "Multi-value-independent ciphertext-policy attribute based encryption with fast keyword search," *IEEE Transactions on Services Computing*, vol. 13, no. 6, pp. 1142–1151, 2020.

[11] L. Cao, Y. Kang, Q. Wu, R. Wu, X. Guo, and T. Feng, "Searchable encryption cloud storage with dynamic data update to support efficient policy hiding," *China Communications*, vol. 17, no. 6, pp. 153–163, 2020.

[12] S. Wang, D. Zhang, Y. Zhang, and L. Liu, "Efficiently revocable and searchable attribute-based encryption scheme for mobile cloud storage," *IEEE Access*, vol. 6, pp. 30444–30457, 2018.

[13] D. Xiaoding Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pp. 44–55, IEEE, Berkeley, CA, USA, 14 May 2000.

[14] E. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security. CCS '12*, pp. 965–976, Association for Computing Machinery, Raleigh, North Carolina, USA, 16 October 2012.

[15] K. Kurosawa and Y. Ohtaki, "UC-secure searchable symmetric encryption," in *Financial Cryptography and Data*

Security, A. D. Keromytis, Ed., pp. 285–298, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[16] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography and Data Security*, A.-R. Sadeghi, Ed., pp. 258–274, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[17] G. Asharov, M. Naor, G. Segev, and I. shahaf, "Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations," in *Proceedings Of the Forty-Eighth Annual ACM Symposium On Theory Of Computing. STOC '16*, pp. 1101–1114, Association for Computing Machinery, Cambridge, MA, USA, 2016.

[18] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds., pp. 506–522, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[19] M. Abdalla, M. Bellare, D. Catalano et al., "Searchable encryption revisited: consistency properties, relation to anonymous IBE, and extensions," in *Advances in Cryptology – CRYPTO 2005*, V. Shoup, Ed., pp. 205–222, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[20] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith, "Public key encryption that allows PIR queries," in *Advances in Cryptology - CRYPTO 2007*, A. Menezes, Ed., pp. 50–67, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[21] F. Bao, R. H. Deng, X. Ding, and Y. Yang, "Private query on encrypted data in multiuser settings," in *Information Security Practice and Experience*, L. Chen, Yi Mu, and W. Susilo, Eds., pp. 71–85, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[22] Y. Liang, Y. Li, Q. Cao, and F. Ren, "VPAMS: verifiable and practical attributebased multi-keyword search over encrypted cloud data," *Journal of Systems Architecture*, vol. 108, pp. 1383–7621, Article ID 101741, 2020.

[23] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1187–1198, 2016.

[24] Y. Miao, R. H. Deng, and X. Liu, "Multi-Authority attribute-based keyword search over encrypted cloud data," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 4, pp. 1667–1680, 2021.

[25] J. Li, M. Wang, Y. Lu, Y. Zhang, and H. Wang, "ABKSSKGA: attribute-based keyword search secure against keyword guessing attack," *Computer Standards & Interfaces*, vol. 74, pp. 0920–5489, Article ID 103471, 2021.

[26] S. Wang, S. Jia, and Y. Zhang, "Verifiable and multi-keyword searchable attribute-based encryption scheme for cloud storage," *IEEE Access*, vol. 7, pp. 50136–50147, 2019.

[27] J. Shi, J. Lai, Y. Li, R. H. Deng, and J. Weng, "Authorized keyword search on encrypted data," in *Computer Security - ESORICS 2014*, M. law Kuty lowski and J. Vaidya, Eds., pp. 419–435, Springer International Publishing, Cham, 2014.

[28] Q. Zheng, S. Xu, and G. Ateniese, "VABKS: verifiable attribute-based keyword search over outsourced encrypted data," in *Proceedings of the IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pp. 522–530, IEEE, Toronto, ON, Canada, 27 April 2014.

[29] R. Bobba, H. Khurana, and M. Prabhakaran, "Attribute-sets: a practically motivated enhancement to attribute-based encryption," in *Computer Security – ESORICS 2009*, M. Backes and P. Ning, Eds., pp. 587–604, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[30] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology – CRYPTO 2001*, J. Kilian, Ed., pp. 213–229, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[31] V. Goyal, O. Pandey, S. Amit, and W. Brent, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security. CCS '06*, pp. 89–98, Association for Computing Machinery, Alexandria, Virginia, USA, 30 October 2006.