Research Article

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

Yu Lin, Lingling Xu, Wanhua Li, and Zhiwei Sun

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

Correspondences 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

Copyright © 2021 Yu Lin et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

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 Medicine, 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 ∧ Doctor name: Mike) ∨ (Drug name: Aspirin ∨ 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 (ASBBKS). 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
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 \( e: G_1 \times G_1 \rightarrow G_2 \) satisfies the following conditions, we say it is a bilinear mapping [30]:

1. Bilinear: for any \( g, h \in G_1 \) and \( a, b \in \mathbb{Z}_p \),
   \[ e(g^a, h^b) = e(g, h)^{ab}. \]

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 associated with an 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 \( \mathcal{A} = \{A_0, A_1, \ldots, A_n\} \) where \( A_0 \) is the set of attributes at depth 1 and \( A_i \) is the set at depth 2 for \( 1 \leq i \leq n \).

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 \( \text{att}(x) \). Furthermore, the parent of node \( x \) is presented as \( \text{parent}(x) \), the label associated with \( x \) is presented as \( \text{label}(x) \), the set of leaf nodes of \( \mathcal{T} \) is presented as \( \text{lvs}(\mathcal{T}) \), and the subtree of \( \mathcal{T} \) rooted at the node \( x \) is presented as \( \mathcal{T}_x \).

For an attribute set structure \( \mathcal{A} = \{A_0, A_1, \ldots, A_n\} \) with depth 2 and an access tree \( \mathcal{T} \), if at least one of the following conditions holds, we say that \( \mathcal{A} \) satisfies \( \mathcal{T} \): (1) there is at least one subset of \( \mathcal{A} \), in which the combination of its attributes satisfies \( \mathcal{T} \); (2) there are some translating nodes where the attributes from multiple sets in \( \mathcal{A} \) 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 \( \mathcal{A} \) 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 \( \mathcal{A} \) 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 \( q_x \).
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) Setup \( (k) \rightarrow \{\text{pk}, \text{mk}\} \): given the security parameter \( k \), the Setup algorithm generates a public key \( \text{pk} \) and a master key \( \text{mk} \).

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

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

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

(v) Test \( (C, T) \rightarrow \{0, 1\} \): 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 "\( \bot \)."

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.
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 (IND-SCKA). In the following game, adversary $A$ interacts with the game challenger $C$ as follows.

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

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

1. **Secret key queries:** adversary $A$ adaptively queries secret keys for recursive attribute set structures to challenger $C$.

2. **Trapdoor queries:** adversary $A$ adaptively queries Boolean keyword value expressions $B_\ell$ to challenger $C$ and gets the corresponding trapdoors.

(iii) **Challenge:** after the above queries, adversary $A$ submits two distinct keyword value sets $W_{V_0}$ and $W_{V_1}$ to challenger $C$. These sets cannot satisfy the Boolean keyword value expression $B_\ell$ that is queried with the form of Trapdoor queries in Phase 1. Then, $C$ randomly selects a bit $\beta \in \{0, 1\}$, constructs the challenge ciphertext $C^*$ of $W_{V_{\beta}}$, and returns it to $A$.

(iv) **Phase 2:** adversary $A$ continues to make secret key queries and trapdoor queries similar to phase 1. It requires both $W_{V_0}$ and $W_{V_1}$ to not satisfy the Boolean keyword value expression $B_\ell$.

(v) **Guess:** adversary $A$ 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** \((\kappa) \rightarrow \{pk, mk\} \). Let $e: G \times G \rightarrow G_T$ 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 $H_i: \{0, 1\}^* \rightarrow Z_p^*$ and $H_i: \{0, 1\}^* \rightarrow G$ 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 $(pk, mk)$ are set to be

\[ pk = \langle G, G_T, p, g, e, h_1, h_2, h_3, H_0 (\cdot), H_1 (\cdot) \rangle, \]
\[ mk = \langle \beta_1, \beta_2, \alpha \rangle. \]  

3.1.2. **KeyGen** \((\mathcal{A}, mk) \rightarrow \{sk\} \). The input recursive attribute set structure is parsed as $\mathcal{A} = \{A_0, A_1, \ldots, A_n\}$. Assume that each $A_i$ has $m_i$ attribute members; then, there is $A_i = \{a_{i,j} \ldots a_{i,m_i}\}$ for $0 \leq i \leq n$. Firstly, it randomly selects a value $r \in Z_p^*$ for attribute set structure $\mathcal{A}$. For each subset $A_i$, it randomly selects $n$ values $\{r_i \in Z_p^* | 1 \leq i \leq n\}$ and sets $r_0 = r$. Furthermore, it randomly selects a set of values $\{r_{i,j} \in Z_p^* | 0 \leq i \leq n, 1 \leq j \leq m_i\}$ for each attribute. Finally, it calculates $B = g^{(\alpha - r)/\beta_i}, B_{i,j} = g^{r_i H_i (a_{i,j})/\beta_i}$ and $B_{i,j} = g^{r_{i,j}}$ for $0 \leq i \leq n, 1 \leq j \leq m_i$ and $D_1 = g^{(r + r_i)/\beta_i}$ for $1 \leq i \leq n$. The secret key for $\mathcal{A}$ is set to be

\[ sk = \langle \mathcal{A}, B, \{B_{i,j}, B_{i,j}^{s_i} \} | 0 \leq i \leq n, 1 \leq j \leq m_i \rangle, \]
\[ \{D_1 | 1 \leq i \leq n\}. \]

At the translating node, each element $D_1$ 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 $D_0$ and $D_1$ can be combined as $D_0/D_1$ to translate $r_i$ to $r_i$ at the translating nodes.

3.1.3. **Encrypt** \((pk, W_V, \mathcal{F}) \rightarrow \{C\} \). $W_V = \{w_1, w_2, \ldots, w_{m+1}\}$ 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 $C_1 = h_1, C_2 = h_2, C_{i,1} = g^{s_i}, C_{i,2} = h_3^{s_i}, h_2^{H_2 (w_{\rho (i)})h_3}, h_2^{h_3}$ for $
1 \leq i \leq m$. After that, it computes secret shares of $s_0$ through implementing \{\(q_i(0) \mid v \in \text{lvs}(\mathcal{T})\)\} \rightarrow \text{Share}(\mathcal{S}, s_0)$. Further, for each $\forall v \in \text{lvs}(\mathcal{T})$, it computes $C_v = g^{q_v(0)}$ and $C'_v = H_i(\text{att}(v))^{q_v(0)}$. We assume that the set of translating nodes in $\mathcal{T}$ is represented as $\text{trans}(\mathcal{T})$. Then, it calculates $\overline{C}_x = h_{\overline{t}}(0)$ for each $x \in \text{trans}(\mathcal{T})$. Finally, the ciphertext is set to be

\[
C = \langle \mathcal{T}, W_{N_1}, C_{1}, C_{2}, \{[C_{1,i}, C_{2,i}] \mid 1 \leq i \leq m\}, \{[C_v, C'_v] \mid v \in \text{lvs}(\mathcal{T})\}, \{\overline{C}_x \mid x \in \text{trans}(\mathcal{T})\} \rangle.
\]

In user keys, elements $(\overline{C}_x)$'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 $(sk, \mathcal{B}_v) \rightarrow \{T\}$. $\mathcal{B}_v$ represents the Boolean keyword value expression which is an access tree. $\mathcal{B}_v$ represents the Boolean keyword name expression which is an access tree with the same structure as $\mathcal{B}_v$. Taking sk and $\mathcal{B}_v$ as inputs, it randomly selects a value $t \in \mathbb{Z}_p$ and calculates the secret shares of $t$ by implementing \{\(q_i(0) \mid v \in \text{lvs}(\mathcal{B}_v)\)\} \rightarrow \text{Share}(\mathcal{S}_N, t)$. Then, this algorithm calculates $\tau_{(v,1)} = (h_1 h_2 (q_2 h_2 + q_v h_0 + q_v') h_0)^t$ and $\tau_{(v,2)} = g^{q_v(0)}$. Parsing sk as \(\{A, B, \{B_{ij} \mid 0 \leq i \leq n, 1 \leq j \leq m_i\}, \{D_i \mid 1 \leq i \leq n\}\}$, it further computes $\overline{B} = B', B_{ij} = B_{ij}', B_{ij}' = B_{ij}, B_i' = B_i$ for $0 \leq i \leq n, 1 \leq j \leq m_i$ and $D_i = D_i'$ for $1 \leq i \leq n$. Finally, the trapdoor for $\mathcal{B}_v$ is

\[
\text{T} = \langle \mathcal{A}, \mathcal{B}_v, \tau_{(v,1)}, \tau_{(v,2)} \mid v \in \text{lvs}(\mathcal{B}_v)\rangle, \overline{B}, \{[\overline{B}_{ij}, B_{ij}'] \mid 0 \leq i \leq n, 1 \leq j \leq m_i\}, \{[D_i] \mid 1 \leq i \leq n\} \rangle.
\]

3.1.5. Test $(C, T) \rightarrow \{0, 1\}$. Cloud server takes input ciphertext $C = \langle \mathcal{T}, W_{N_1}, C_{1}, C_{2}, \{[C_{1,i}, C_{2,i}] \mid 1 \leq i \leq m\}, \{C_v, C'_v\} \mid v \in \mathcal{V}\rangle, \{\overline{C}_x \mid x \in \text{trans}(\mathcal{T})\rangle$ and the trapdoor $T = \langle \mathcal{A}, \mathcal{B}_v, \tau_{(v,1)}, \tau_{(v,2)} \mid v \in \text{lvs}(\mathcal{B}_v)\rangle, \overline{B}, \{[\overline{B}_{ij}, B_{ij}'] \mid 0 \leq i \leq n, 1 \leq j \leq m_i\}, \{[D_i] \mid 1 \leq i \leq n\} \rangle$. 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 $\mathcal{A}$, it returns a set $\mathcal{S}_T$. The elements of this set are some labels for each node $r \in \mathcal{T}$. Each label $t \in \mathcal{S}_T$ represents a set $A_t$ and each set satisfies the subtree $\mathcal{T}_r$. There is $\mathcal{T}_r = \mathcal{T}$ for the root node $R$ and the related set is $\mathcal{S}_R$. If $\mathcal{A}$ does not satisfy $\mathcal{T}$, the return of this algorithm is “0.” Otherwise, for node $r$, it picks one label from the set $\mathcal{S}_T$, denoted as $i$. Then, it executes a recursive function DecryptNode($C, T, r, i$), which will return $F_r$ as the result. According to the type of node $r$, the function DecryptNode($C, T, r, i$) will be computed in two different ways.

(1) When $r$ is a leaf node: if $\text{att}(r) \notin A_i$, it returns “1.” Otherwise, we have

\[
\text{DecryptNode}(C, T, r, i) = \frac{e(C_r, B_{ij})}{e(C'_r, B_{ij}')} = e(g, g)^{t \cdot q_v(0)}. \tag{5}
\]

(2) When $r$ is not a leaf node:

(i) Firstly, it calculates a set $E_r$, which is composed of $k_r$ child nodes of $r$. In $E_r$, each node $z$ must satisfy one of the following two cases: (1) label $i \in E_z$, and (2) $z$ is a translating node and there exists a label $i'$ that satisfies $i' \in E_z$ and $i' \neq i$. If such a set does not exist, it returns “1.”

(ii) Execute DecryptNode($C, T, z, i'$) for each translating node $z \in E_r$ which satisfies label $i \in E_z$ and return $F_z$ as the result.

(iii) Execute DecryptNode($C, T, z, i'$) for each translating node $z \in E_r$ which satisfies $i' \in E_z$ and $i' \neq i$ and return $F'_z$ as the result.

If $i = 0$, elements $\overline{D}_1$ and $\overline{C}_z$ can be used to translate $F'_z$ to $F_z$.

\[
F_z = \frac{e(\overline{D}_1, \overline{C}_z)}{F'_z} \cdot e(g, g)^{r \cdot q_v(0)} = e(g, g)^{r \cdot q_v(0)}. \tag{6}
\]

If $i \neq 0$, elements $\overline{D}_1$ and $\overline{D}_i$ together with $\overline{C}_z$ can be used to translate $F'_z$ to $F_z$.

\[
F_z = e(\overline{D}_1, \overline{C}_z), F'_z \cdot \frac{e(g, g)^{r \cdot q_v(0)}}{e(g, g)^{r \cdot q_v(0)}} = e(g, g)^{r \cdot q_v(0)}. \tag{7}
\]

(iv) After computing $F_z$ for each node $z \in E_z$, it computes $F_r$ as follows:

\[
F_r = \prod_{z \in E_z} F_z \Delta_{z, A_z} = \begin{cases} e(g, g)^{t \cdot q_v(0)}, & i = 0, \\ e(g, g)^{r \cdot q_v(0)}, & i \neq 0 \end{cases}, \tag{8}
\]

where $\Delta_{z, A_z}(x) = \prod_{j \in z, j \neq x} (x - j)/(i - j), \forall = \text{label}(z)$, and $U_z = \{\text{label}(z) \mid z \in E_z\}$.

After the above steps, it further computes the function DecryptNode($C, T, R, i$) for root node $R$ and returns $F_R$ as the result as follows.
\[
F_R = \begin{cases}
   e (g, g)^{r_{i=0}}, & i = 0, \\
   e (g, g)^{r_{i \neq 0}}, & i \neq 0,
\end{cases}
\]  

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:
\[
F = e (D_i, C_s) = e (g, g)^{r_{i+1}} = e (g, g)^{r_{i}} = e (g, g)^{r_{i-1}}. 
\]  

**Step 2.** Firstly, it computes a value \( L \) with the following formula:
\[
L = e (\overline{B}, C_i) = e (g, g)^{(a-r)\beta_i, g^{\beta_i}} = e (g, g)^{(a-r)\beta_i}. 
\]  

Then, it computes \( F \cdot L \) with the following formula:
\[
F \cdot L = e (g, g)^{r_{i}} e (g, g)^{r_{i}} = e (g, g)^{r_{i}}. 
\]

**Step 3.** It selects a minimum subset \( W_N \) from the set of keyword names \( W_N \), which satisfies the Boolean keyword name expression \( \mathcal{B}_N \). If \( W_N \) not exist, it returns “0.” If node \( \overline{x} \) is a leaf node of access tree \( \mathcal{B}_N \) and is related to the search token \( T \), the keyword name associated with this node is denoted by \( \rho (\overline{x}) \). Further, for each keyword name \( \rho (\overline{x}) \in W_N \), it computes
\[
E_{\overline{x}} = e (T_{\rho (\overline{x})}, C_{\rho (\overline{x})}) = e (g, g)^{(a-r)\beta_i} = e (g, g)^{(a-r)\beta_i}. 
\]

If \( \omega_{\rho (\overline{x})} = g^{\rho (\overline{x})} \), then there is \( E_{\overline{x}} = e (g, g)^{a\omega_{\rho (\overline{x})}} \).

When node \( \overline{x} \) is not a leaf node, for all child nodes \( \overline{z} \) of \( \overline{x} \), assume that \( \mathcal{B}_N \) represents an arbitrary set with size \( k_{\overline{z}} \) consisting of children nodes \( \overline{z} \) and \( E_{\overline{z}} \neq 1 \). If \( S_{\overline{x}} \) does not exist, then \( E_{\overline{x}} \neq 1 \); otherwise, it utilizes the polynomial interpolation to calculate \( E_{\overline{x}} \) to get
\[
E_{\overline{x}} = e (g, g)^{a\omega_{\rho (\overline{x})}}. 
\]

At last, for the root node of \( \mathcal{B}_N \), it computes \( E_{\overline{z}} \) and checks whether the following equation holds:
\[
E_{\overline{z}} = F \cdot L. 
\]  

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 \( \beta_i \) needs to be selected. \( \beta_i \) 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 \( \beta_i \).

### 3.2. Security Proof

**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 \( \mathcal{A} \) assumes to distinguish \( g^{a(s+i)}g^{\beta_iH_i(w^{\rho (\overline{x})})} \) from \( g^{\theta} \) and \( g^{a(s+i)}g^{\beta_iH_i(w^{\rho (\overline{x})})} \) from \( g^{\theta} \), where \( \theta \) is randomly selected from \( Z^*_p \). \( W_0 \) and \( W_1 \) are two different keyword sets, \( w^{\rho (\overline{x})} \in W_0 \) and \( w^{\rho (\overline{x})} \in W_1 \). Since \( \mathcal{A} \) has the same probability to distinguish both of them, it is easy to distinguish \( g^{\beta_iH_i(w^{\rho (\overline{x})})} \) from \( g^{\theta} \). That is, this game can be transformed to \( \mathcal{A} \) that has the advantage \( e/2 \) to distinguish \( g^{a(s+i)} \) from \( g^{\theta} \).

The challenger \( \mathcal{C} \) can be constructed in this game as follows.

(i) **Setup:** the challenger \( \mathcal{C} \) selects parameters \( \alpha, \beta_i, \beta_i \in Z^*_p \) and sends public parameter \( pk = (G, G_T, p, g, e, g^{\beta_i}, g^{\beta_i}, H_A) \) to \( \mathcal{A} \). After that, \( \mathcal{A} \) selects a challenge access tree \( T^* \) and returns it to \( \mathcal{C} \).

(ii) **Hash/Query:** challenger \( \mathcal{C} \) 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 \( g^{\theta} \) to \( \mathcal{A} \) 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, \( \mathcal{C} \) will directly return \( g^{\theta} \) to \( \mathcal{A} \). \( \mathcal{A} \) can query this random oracle for polynomially many times.

**Phase 1.** Adversary \( \mathcal{A} \) can make the secret key and trapdoor queries for polynomially many times.

(i) **Secret key queries:** firstly, for an attribute set structure \( A = \{A_0, A_1, \ldots, A_n\} \) where \( A_i = \{a_{i,1}, \ldots, a_{i,m_i}\} \) for \( 0 \leq i \leq n \), \( \mathcal{C} \) will randomly select a value \( r \in Z^*_p \) for \( A_i \) and compute \( B = g^{(a-r)\beta_i} \). For each subset \( A_i \), it will randomly select \( t_{i,j} \in Z^*_p \) and give a value \( r_{i,j} \in Z^*_p \) for \( 0 \leq i \leq n, 1 \leq j \leq m_i \). Next, these parameters to compute corresponding \( B_{i,j} = g^{t_{i,j}}, B_{i,j} = g^{r_{i,j}} \), and \( D_{i,j} = g^{(s+i)\beta_i} \). Finally, \( \mathcal{C} \) constructs the secret key sk = \( \{A_i, B, (B_{i,j})_{0 \leq i \leq n, 1 \leq j \leq m_i}, D, 1 \leq i \leq n\} \) with the above parameters and returns it to \( \mathcal{A} \).

(ii) **Trapdoor queries:** if \( \mathcal{A} \) has queried secret key sk, then \( \mathcal{C} \) directly runs the Trapdoor algorithm to get
the trapdoor $T$. Challenger $\mathcal{C}$ randomly selects $t \in \mathbb{Z}_p^*$ and computes the secret shares of $t$ by running $\{\tilde{q}_i(0) \mid v \in \text{Ivs}(\mathcal{B}_N)\} \leftarrow \text{Share}(\mathcal{B}_N, t)$ and computes corresponding $T_{p(v),1} = (g^a g^{Bv} (\mathcal{w}_{p(v)})^{\tilde{v}(0)})$, $T_{p(v),2} = g^{\tilde{v}(0)}$, $B = B_i, B_{i,j} = B_i + B_j$, $T_{i,j} = D_i$. Then the trapdoor for Boolean keyword value expression $\mathcal{B}_v$ is $T = \langle \mathcal{B}_v \rangle \{(T_{p(v),1}, T_{p(v),2}) \mid v \in \text{Ivs}(\mathcal{B}_N), \mathcal{B}_i \{(B_{i,j}) \mid 0 \leq i \leq n, 1 \leq j \leq m\}, \{T_{i,j} \mid 1 \leq i \leq n\}\}$, and the challenger $C$ returns it to $\mathcal{A}$.

**Challenge:** $\mathcal{A}$ submits two distinct keyword value sets $W_{V_0} = (w_{p(1)}, w_{p(2)}, \ldots, w_{p(m)})$ and $W_{V_1} = (w_{p(1)}, w_{p(2)}, \ldots, w_{p(m)})$ to $\mathcal{C}$, and these sets are of equal size. Then, let $W_N = (\rho(1), \rho(2), \ldots, \rho(m))$ represent the set of keyword names. For these two sets, they cannot satisfy the Boolean keyword value expression $B_v$. Then, $\mathcal{C}$ randomly selects $s_0, s_1 \in \mathbb{Z}_p^*$ for $1 \leq i \leq m$ and computes secret shares of $s_0$ by executing $\text{Share}(\mathcal{T}^*, s_0) \rightarrow \{\tilde{q}_i(0) \mid v \in \text{Ivs}(\mathcal{T}^*)\}$.

Then, $C$ randomly selects a bit $\beta \in \{0, 1\}$ and constructs the challenge ciphertext $C^* = \langle \mathcal{T}^*, W_N, C_1, C_2, \{(C_{i,1}, C_{i,2}) \mid 1 \leq i \leq m\}, \{(C_v, C_v') \mid v \in \text{Ivs}(\mathcal{T}^*), C_v = g^{Bv}, C_v' = g^{Bv'}, \text{Ivs}(\mathcal{T}^*), \text{att}(v) = a_{i,j}\}, \{C_{x} \mid 0 \leq x \leq \mathcal{N} \rangle$.

(i) If $\beta = 0$, it randomly selects $\theta \in \mathbb{Z}_p^*$ and outputs $C_1 = g^{Bv}, C_2 = g^{Bv'}, \text{Ivs}(\mathcal{T}^*), \text{att}(v) = a_{i,j}$, then the challenger $C$ returns it to $\mathcal{A}$.

(ii) If $\beta = 1$, it outputs $C_1 = g^{Bv}, C_2 = g^{Bv'}, \text{Ivs}(\mathcal{T}^*), \text{att}(v) = a_{i,j}$, then the challenger $C$ returns it to $\mathcal{A}$.

**Phase 2.** After receiving the challenge ciphertext $C^*$, $\mathcal{A}$ can perform secret key queries and trapdoor queries mentioned in phase 1. It requires both $W_{V_0}$ and $W_{V_1}$ to not satisfy the Boolean keyword value expression $B_v$.

**Guess:** $\mathcal{A}$ outputs its guess, which requires $\mathcal{A}$ to distinguish $g^{Bv}$ from $g^{Bv'}$ to win this game.

**Analysis:** if $\mathcal{A}$ can construct $e(g^{Bv'}, g^{Bv})$ for some $\mathbf{g}^\alpha$, $\mathcal{A}$ has the ability to distinguish $g^{Bv'}$ from a random element $g^\alpha$. Thus, we continue to analyze that adversary $\mathcal{A}$ constructs $e(g, g)^{\eta \mathbf{g}^{(s_i, t_{i+1})}}$ for some $\mathbf{g}^\alpha$ with negligible advantage. That means $\mathcal{A}$ cannot win our game with a non-negligible advantage.

From the above phases, it can be easily found that $\mathcal{s}_i$ and $\mathcal{A}$ have appeared, so $\mathcal{A}$ only needs to construct $\mathcal{a}_\mathcal{s}_0$. With
terms $g^{r_{i0}}$ and $g^{(a+r_{i0})}$, $A$ can construct $e(g, g)^{a r_{i0}} e(g, g)^{r_{i0}}$ and then $A$ needs to conceal $e(g, g)^{r_{i0}}$. With terms $g^{r_{i0} r_{j0}}$ and $g^{(r_{i0} r_{j0})}$, $A$ can construct $e(g, g)^{(a + r_{i0} r_{j0})}$ and $A$ needs to conceal $e(g, g)^{r_{i0} r_{j0}}$. Then, $g^{r_{i0} r_{j0}}$ should be constructed which will use terms $g^{r_{i0} r_{j0} r_{k0}}$, $g^{(r_{i0} r_{j0} r_{k0})}$, and $g^{r_{i0} r_{j0} r_{k0}}$ because $q(0)$ is the secret share of $s_0$. However, $g^{r_{i0} r_{j0} r_{k0}}$ cannot be constructed since there are not enough attribute values $g^{q(0)}$ for $A$ to satisfy the access tree $F$.

### 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 $A = \{A_0, A_1, \ldots, A_l\}$ where $A_i = \{a_{i1}, a_{i2}, \ldots, a_{il}\}$. Let $M = m_0 + m_1 + \cdots + m_l$. Then, let the leaf nodes of an access tree $F$ be $\text{lvs}(F)$ and translating nodes be $\text{trans}(F)$. 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 $B_k$ to be $\text{lvs}(B_k)$. 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 $|\text{trans}(F)| + 2|\text{lvs}(F)| + 3N + 2$ exponentiations in $G$. The storage overhead is $|\text{trans}(F)| + 2|\text{lvs}(F)| + 2N + 2$ group elements in $G$. In [28], the Encrypt algorithm takes $2|\text{lvs}(F)| + 4$ exponentiations in $G$. The storage overhead is $2|\text{lvs}(F)| + 3$ group elements in $G$.

### Table 1: Comparison of functionalities between SE schemes.

<table>
<thead>
<tr>
<th>Boolean keyword search</th>
<th>Access structure</th>
<th>Compound attributes</th>
<th>Translating nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>√</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>[28]</td>
<td>×</td>
<td>Access tree</td>
<td>×</td>
</tr>
<tr>
<td>ASBBKS</td>
<td>√</td>
<td>Access tree</td>
<td>√</td>
</tr>
</tbody>
</table>

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

**Trapdoor.** For the Trapdoor algorithm in our scheme, its computation complexity is $2M + n + 3|\text{lvs}(B_N)| + 1$ exponentiations in $G$. The storage overhead is $2M + n + 1 + 2|\text{lvs}(B_N)|$ 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 $1 + 2M + 3|\text{lvs}(B_N)|$ exponentiations in $G$. The storage takes $1 + 2M + 2|\text{lvs}(B_N)|$ group elements in $G$.

**Test.** For the Test algorithm in our scheme, we first assume $A_i$ has $l$ leaf nodes satisfying $F$ and $k_i$ translating nodes on the path from $i$th leaf node used to the root node where $1 \leq i \leq l$, and let $k = k_1 + k_2 + \cdots + k_l$. In addition, we denote the number of all used nodes in $F$ as $t$, and the computation complexity is $2l + k + 1 + 2|\text{lvs}(B_N)|$ pairings and $t - 1 + |\text{lvs}(B_N)|$ 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 $1 + 2l + 2|\text{lvs}(B_N)|$ pairings and $|\text{lvs}(F)| + |\text{lvs}(B_N)|$ 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 $A$, and the number of translating nodes in $F$. When there is only one set in $A$, one keyword in the Encrypt algorithm and Trapdoor algorithm, and no translating node in access tree $F$, 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>
<thead>
<tr>
<th></th>
<th>[31]</th>
<th>[28]</th>
<th>[11]</th>
<th>Our scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computation costs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KeyGen</td>
<td>$(2M + 1)E$</td>
<td>$(1 + 2n + 2M)E$</td>
<td>$(2 + 2n)E$</td>
<td>$(1 + 2n + 2M)E$</td>
</tr>
<tr>
<td>Encrypt</td>
<td>$(2</td>
<td>\text{trans}(T)</td>
<td>+ 4)E$</td>
<td>$(</td>
</tr>
<tr>
<td>Trapdoor</td>
<td>$(2M + 4)E$</td>
<td>$(4 + n + 2M)E$</td>
<td>$(1 + 2M + 3</td>
<td>\text{vs}(B_N)</td>
</tr>
<tr>
<td>Test</td>
<td>$(2</td>
<td>lvs</td>
<td>+ (t - 1)E_T)E_T$</td>
<td>$(2</td>
</tr>
<tr>
<td><strong>Communication costs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KeyGen</td>
<td>$(2M)G$</td>
<td>$(1 + n + 2M)G$</td>
<td>$(1 + 2M)G$</td>
<td>$(1 + n + 2M)G$</td>
</tr>
<tr>
<td>Encrypt</td>
<td>$(2</td>
<td>\text{vs}(T)</td>
<td>+ 3)G$</td>
<td>$(2</td>
</tr>
<tr>
<td>Trapdoor</td>
<td>$(2M + 3)G$</td>
<td>$(3 + n + 2M)G$</td>
<td>$(1 + 2M + 2</td>
<td>\text{vs}(B_N)</td>
</tr>
<tr>
<td>Test</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
</tbody>
</table>

* 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 $\mathcal{A}$; $n$: number of sets in $\mathcal{A}$; $l$: number of attributes used in $\mathcal{A}$ required to satisfy $T$; $t$: number of all used nodes in $T$; $|\text{vs}(T)|$: number of leaf nodes used in $T$; $|\text{vs}(B_N)|$: 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.
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 $A$ and the number of translating nodes in $T$. When there is one set in $A$ and no translating node in access tree $T$, 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 $A$ 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 (20210301001005) and Shenzhen Science and Technology Program (JCYJ20210324100813034).

References


