Efficient Retrieval Over Documents Encrypted By Attributes in Cloud Computing

Dipak Vare, Akshay Kapadi, Deepak Ratnaparkhi, Vaibhav Kale
Prof. Sunil S. Khatal (Head of Department, Computer Engineering)
Sharadchandra Pawar College of Engineering, Pune

Abstract— secure data storage and retrieval is the best research directions in cloud. Though lots of searchable encryption scheme have been proposed some of them support efficient retrieval over the documents. Which are encrypted based on their attributes. In this paper a hierarchical attribute based encryption scheme is designed for a data collection. A set of documents is encrypted together if they share an integrated access structure. Compared with the CP policy attribute based encryption schemes, both the cipher text storage space and time costs of encryption and decryption are saved. Then, an index structure named attribute based retrieval features tree is constructed for the document collection based on the TF-IDF model and the documents attributes. A depth first search algorithm for the attribute based retrieval features tree is designed to better the search efficiency which can be further improved by parallel computing. Except for the documents collections in our scheme can be applied to other data sets by modifying the attribute based retrieval features tree slightly. A thorough analysis and series of experiments performed to illustrate the security and efficiency of the proposed scheme.

Index Terms-Cloud, Document retrieval, file hierarchy, attribute-based encryption.

I. INTRODUCTION

Lots of people and organizations are motivated to outsource their local document management systems to the cloud which is a promising information technique to process the explosive expanding of data. Cloud computing can collect and reorganize a huge amount of IT resources and evidently, the cloud servers can provide more secure, flexible, various, economic and customize services compared with the local management systems. For all the advantages of cloud services, leaking the sensitive information, such as personal information, company financial data and government documents to the public is a big threat to the data owners. In addition to make full use of the documents on the cloud the data users has to access them flexibly and efficiently. Consequently, a big challenge of outsourcing the data to the cloud is how to protect the confidentiality of the data properly while maintaining their search ability.

An instinctual approach is encrypt the data first and then outsourcing the encrypted data to the cloud. A large number of searchable data encryption scheme have been proposed in the literatures including single keyword Boolean search scheme single keyword ranked search schemes and multi keyword Boolean search schemes. However, all these schemes cannot support effective, flexible and efficient data search because of their simple functionalities, Privacy-preserving multi-keyword ranked document search schemes are more promising and Practical. However, all the data in these scheme are organized by a coarse grained access control mechanism that is each permitted data user can access all the encrypted data. As an Example, the whole IEEE Explore Digital Library can be accessed by all the authorized organizations (e.g. the universities, school) at present and this can't satisfy the data owners and users in the future.

In this paper, a new circumstance is considered. A data user may be want to access part of the library (e.g. computers and data related papers etc.) and intuitively she wants to pay less money compare with the data users who want to access the whole library. In different words, in the data collection, each document can be accessed only by a set of specific data users. In this case, we need to design a fine grained access control mechanism for the data and it is more reasonable compared with the current method.

To make the data users able to access part of IEEE Explore Digital Library on demands, a possible approach is encrypting the documents through attribute-based encryption (ABE) schemes before outsourcing them to the cloud. Meanwhile, the permitted data users are assigned with a set of attributes. A data user can decrypt file if and only if her attributes match the files attributes. Recently, cipher text- policy attribute-based encryption (CP-ABE) is a hot research area and it can provide fine-grained, one to many and flexible access control. In these scheme each document is encrypted individual and their encryption efficiency can be better by employing hierarchical attribute based encryption schemes. However, these scheme can't be employed directly to solve our problem properly. First, existing schemes focus on encrypting a single access tree.

However, it is impossible that all the documents in IEEE Explore Digital Library share a single access tree and how to construct a set of optimized retrieve trees for the document collection is a big challenge. Second, in most existing schemes, when the documents are mapped to a set of shared retrieve trees, the data users need to store a huge number of secret keys which will be study in Section IV.B. Apparently, this is a heavy burden for the data users especially for an extremely large document collection and how to decrease the amount of secret keys for the data users is another challenge. Except for access control, document search efficiency is also a challenge for a large document collection. To our knowledge, most existing schemes can't support time-efficient retrieval over the documents which are organized under attribute-based access control mechanism.

To support the previously discussed service, we first design an algorithm to generate hierarchical retrieve trees for the document collection. The proposed algorithm take on the greedy strategy to build the access trees incrementally and each access tree grow by continuously splitting the nodes in the tree. Then we design a cipher text policy attribute based hierarchical document collection encryption scheme called CP-ABHE. In the suggested scheme, a set of documents can share a same integrated access tree and be encrypted together rather than being encrypted individually. In this way, both the cipher text storage space and time costs of the encryption/decryption are saved. The security of the proposed scheme is proved theoretically, and its capability is also evaluated by simulation.

To support exact and efficient document search over the encrypted documents, a complicated index structure is then constructed for the document collection. We first map the documents to document vectors based on the TF-IDF model and in addition, the attributes of the documents are also taken into thought. The similar function between the document vectors is thoroughly design and the vector are organize based on their relative similarity in the attribute based retrieval features tree. Specifically, the similar vectors compose micro cluster which are then, aggregated with each other to generate macro clusters until all the vectors belong to one cluster. The attribute based retrieval features vector of the node in the tree are used to describe the inherent properties of the cluster represented by the node. At last a depth first search algorithm for the attribute based retrieval features tree is designed to both the search efficiency and accuracy.

The main contributions of this paper are summarized as follows:

• A practical hierarchical attribute-based document muster encryption scheme is proposed in which the documents are organized and controlled based on attributes. The proposed scheme can greatly decrease the storage and computing load.

• We map the documents to vectors in which both the keywords and associated attributes are considered. The ARF tree is proposed to organize the document vectors and support time-efficient document accessible. In addition, a depth-first search algorithm is designed.

• A partial simulation is performed to illustrate the security, efficiency and effectiveness of our scheme. Specifically,

The proposed encryption scheme perform well in both time and storage efficiency. In addition, our scheme also provides efficient and accurate data retrieval method.

The left of this paper is organized as follows, the related work is provided in Section II and Section III, and we stated the problem and present some preliminary techniques. The hierarchical attribute based data encryption scheme is designed in Section IV and we present the time efficient document retrieval approach based on the attribute based retrieval features tree in Section V. The security and efficiency analysis of our scheme is provided in Section VI and we further evaluate the performance of the proposed approach in Section VII. At last, Section VIII concludes this paper.

II. Related Work

Our approach is mainly related with two research fields of Cloud, i.e, cipher text-policy attribute-based document encryption and encrypted document retrieval. The related work in these two fields is provided in the following.

Since Sahai et al. proposed the identity based encryption scheme, many ABE schemes have been proposed in which CP-Attribute based encryption schemes are very promising because of their flexibility and scalability. In these CP-Attribute based encryption schemes, the documents with different access structures need to be encrypted individually. To improve the encryption and decryption efficiency and scalability hierarchy attribute based encryption has been widely researched in which a set of documents may share a common access structure and can be encrypted together. Wang et al. propose a hierarchical attribute-based encryption scheme named FH-CP- Attribute based encryption and have proved its security theoretically. An advantage of the scheme is that the data users can decrypt all the authorized documents by the secret key once. Therefore, both the time costs of encryption and decryption are saved. Wang et al. design a scheme named HABE with the traits of high performance, fine- grained access control, scalability and full delegation. HABE is a combination of hierarchical identity-based encryption and CP-Attribute based encryption. Wan et al. propose hierarchical attribute-set-based encryption scheme (HASBE) by extending cipher text-policy attribute-set-based encryption (ASBE) with a hierarchical structure of the data users. The HASBE scheme can seamlessly incorporated with hierarchical structure of system users by applying a delegation algorithm to ASBE. Deng *et al.* extend Attribute based encryption to CP-HABE to support hierarchically distributing and delegating the secret keys which can be used in huge organizations. Guo *et al.* propose a resilient-leakage hierarchical attribute-based encryption scheme to defend against the auxiliary input leakage attack and the security of the scheme is detailed analysed.

In addition to encrypting the document we also attempt to search the encrypted document efficient and accurate. Consequently multi keywords ranked data retrieval over encrypted documents collections is also strong related with our scheme. In Cao *et al.* first proposed a basic privacy preserving multi keyword ranked search scheme based on secure K-Nearest Neighbour algorithm. A set of strict privacy requirements are established and then two scheme are proposed to improve the security and search experience. However, an apparent drawback of this scheme is that the search efficiency is linear with the cardinality of the data collection and consequent it can't be used to process extremely large document databases. Xia *et al.* design a keyword balanced binary tree to organize the document vectors and propose a Greedy Depth First Search algorithm to improve the search efficiency. Moreover the index tree can be updated dynamic with an acceptable communication load. However, the document vectors are chaotically organized in the tree and the search efficiency can be further improved. Chen *et al.* take the relationships of documents into consideration and a hierarchical-clustering-based index framework is designed to improve the search efficiency. In addition a verification scheme is also integrated into their scheme to correctness of the results. Though the index structure can obtain sub linear search efficiency it can't return the accurate search results. Fu *et al.* present a personalized multi keyword ranked search scheme in which an

Interest model of the data users is integrated into the data retrieval system to support personalized search and better users search experience. The interest model of a data user is built based on her search History with the help of word net in order to depict her behaviours in fine grit level. However, this scheme can't support dynamic update operation because the document vector are constructed based on the statistical information of data in the collection. In addition though a MDB tree is employed to improve the search efficiency of the tree is hard to predict Li *et al* propose a new attribute based encryption scheme (KSF-OABE) which can implement keyword search function. Though the design goal of KSF- OABE is some similar with our scheme it cannot hierarchically encrypt a document collection and support efficient multi keyword data retrieval.

III. Problem Statement and Preliminaries

In this section we stated the problems and provide the related preliminary techniques. For convenience and some notations are first defined as follows:

• $F-$ The plaintext document collection of N files, denoted as $F = \{ F_1, F_2, \cdot\cdot, F_N \}$. Each document is treated as a Sequence of keywords. Note that, each file $F_i \leq (1 \; i \leq N)$ has a Unique identifier f_i $(\leq 1 \leq i \; N)$in the whole document collection.

• $A-$ The attribute dictionary, denoted as $A = \{ A_1, A_2, \cdot\cdot, A_n \}$. Each document and data user is associated with a set of attributes in. A

• $C-$ The cipher text off. In this paper is symmetrically encrypted by content secret keys $ck = \{ck_1, ck_2, \cdots, ck_N\}$, i.e., $C_i = E_{cki}(F_i)$, $i = 1, 2, \cdots, N$ and all the cipher texts of the files compose C.

• $I-$ The index structure of F. Each document is mapped to a documents vectors and the vectors are organized in an attribute based retrieval features tree.

• $W-$ The keyword dictionary, denoted as $W = \{ w_1, w_2, \cdot\cdot, w_m \}$ which is used to generated the document vectors and query vectors.

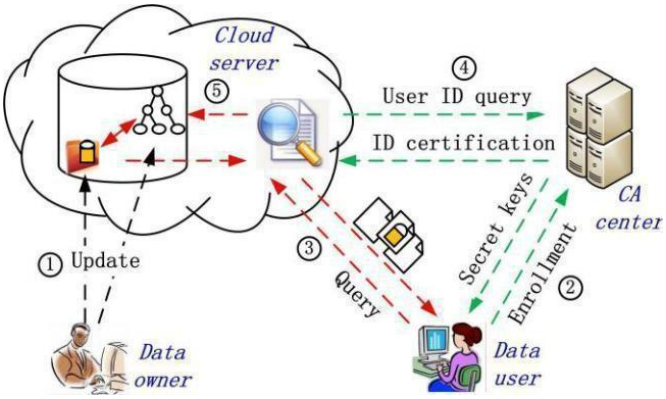

Fig. 1. System model.

• W_Q − A subset of W Represent the keywords in a query.

• $Q-$ The document query request of a user. Each query contains multiple keyword which are describe the interested documents. In addition the attributes of the data user are also added into Q to check the legality of a document. We say that a document has legitimate attributes if the document attribute set is a subset of the data user attribute set and this will be discussed in Section IV.A.

• $SR-$ The result of a search request, i.e. a set of encrypted document which are the top k relevant document to the request under the constraint of a data user attribute.

A. System Model and Design Goals

In this paper, we attempt to design a fine grained access control mechanism for the encrypted document which also support efficient document search. The search result of a query is defined as the top-k relevant encrypted documents with legitimate attributes. The process of executing a document query is presented in Fig. 1 and it is composed of five stages:

The data owner is responsible for collecting and pre-processing the data and then obtains a set of high quality files F He sets the attributes for each document and then hierarchically encrypt the data collection based on attributes. In addition an index vectors is extracted from each document based on the documents content and attributes. An index structure I is constructed based on the index vectors of the document. At final both the encrypted documents and encrypted index structure are sent to the cloud. The cloud server is responsible for store the encrypted data and executing data search based on the index structure.

When a data user wants to search a set of interested document, she first needs to register herself as an authorized data user at the certificate authority (CA) centre. Then, if possible, several attributes selected from Aare assigned to the data user By CA and a corresponding secret key associated with these attributes and sent to the data user. At final the data user can send a query request to the cloud server.

If a query is comes from data-user, the cloud server first communicates with the CA to check the legitimate the data user and her attributes. If the data user is permitted the cloud searches the index structure to obtain the

search result *SR*. Then the corresponding encrypted document are extracted from the encrypted document collection *C* and sent to the data user. At last, the data user decrypts the documents by her secret key. Note that, the legality checking functionality is optional which can be employed to improve the security level of the whole system. With legality checking, the data users who did not register themselves in the CA center cannot search the interested documents through the cloud server. However, the security of the system does not greatly decrease without this functionality and it can be explained by the fact that the illegal data users cannot decrypt the documents returned by the cloud server because they don't have the secret keys.

In this paper, we assume that the CA centre and the cloud are trustable. Specifically, the CA centre can distribute proper attribute to the document users and the cloud server can Execute all the instructions honestly. We further assume that the data users are greedy and attempt to obtain as many Plaintext files as possible. The data users try to collude with Other users to decryption the encrypt documents. We mainly restrict our attention to the process of encryption, document search and decryption, and the design goals of our scheme are presented as follows:

- *Flexibility:* The documents can be encrypted and decrypted flexibly based on their attributes. In general, we hope that the proposed scheme can get logarithmic encryption and decryption time efficiency.
- *Compatibility:* For a data user with an attribute set, she needs to store only one secret key and the key can be used to decrypt all the documents that have legal attributes.
- *Accuracy:* The search results are accurate according to the data users' search request.
- *Efficiency:* Our scheme aims to achieve logarithmic search efficiency over the encrypted files in general and at least sub-linear search efficiency in the worst case.

B. Document/Query Vector

In this paper, the vector of a document is composed of two parts including a normalized content vector and an attribute vector. To build the content vector, each document is treated as a stream of keywords and we use the normalized term frequency (TF) vector to quantize the documents [40]. For each keyword w_i in keyword dictionary W, we denote the number of times that this keyword appears in the document F_j by $f_{j,wi}$ and the TF value of keyword w_i in F_j is defined $= ln(1 + f_{j,w_i})$. We construct the content vector. this vector by

$$TF_{j,w_i} = \frac{\sqrt{f_{j,w_i}}}{\sum_{w \in W} f_{j,w}}, \quad i = 1, 2, \cdots, m$$

At last, the normalized content vector for F_j is denoted as V_j $(=TF_{j,w1}, TF_{j,w2}, \cdots, TF_{j,wm})$. The inverse document frequency (IDF) value of the keyword w_i is defined as $IDF_{wi} = ln(\frac{N}{\text{documents in the}})$ where N is the number of

contain the keyword w_i. Further, the query vector of a query j attributes in the similar manner. At last, we adopt the widely used "TF-IDF" measurement to calculate the relevance score between a document F_j and a query Q as follows:

$$RScore(F_j, Q) = RScore(V_j, V_Q) = V_j \cdot V_Q \quad (2)$$

It can be observed that the attribute vectors are not employed when calculating the relevance scores between a document and a query. This is reasonable considering that we need to return the legal documents of the query rather than the documents that have similar attributes with the query.

C. Attribute-Based Retrieval Feature and ARF Tree

To improve the search efficiency of multi-keywords search process, a height-balanced index tree named ARF tree is built based on the document vectors. Specifically, the document vectors are organized as clusters according to their similarities. Each node in the tree represents a cluster composed of a set of document vectors or sub-clusters. An ARF vector is a quintuple summarization about a cluster. Given P documents F where j 1, 2, , P, we assume that a cluster C comprises the document vectors of { } where $j = 1, 2, \cdots, P$. Then, the ARF vector of the cluster is defined as follows: $ARF = (P, LS, SS, V_{max}, A_{min})$, where P is the number of document content vectors in the cluster, ΣLS is the linear sum of the P content vectors, i.e., $LS = \sum_{j=1}^{P} V1_j$, SS is the square sum of the P content vectors, i.e., $SS = \sum_{Pj=1} V^{j2}$, V denotes a vector consisting of m values which are calculated as follows:

$$V_{max}[i] = max(V_1[i], V_2[i], \cdots, V_P[i]), \quad i = 1, 2, \cdots, m \quad (3)$$

where $V_j[i]$ is the i-th dimensional value of V_j, A_{min} is the common attribute set vector of the documents in the cluster and it can be calculated as follows:

$$[i], \quad i = 1, 2, \cdots, n \quad (4)$$

1 2 P

pair of bits in Vi^* and Vj^*, logic operation "∨" returns 1 if either of the two bits is 1; otherwise, "∨" returns 0. As an example, $(1, 0, 0, 1) \wedge (1, 1, 0, 0) = (1, 0, 0, 0)$; $(1, 0, 0, 1) \vee (1, 1, 0, 0) = (1, 1, 0, 1)$.

In this paper, a search request of a data user contains both a set of keywords $_QW$ and a set of attributes S_U associated with The data user. Only the documents, whose attributes are

matched with S and contents are relevant with Q, are the challenger outputs a random 5-tuple $(g, A = g^a, B =$ returned to the data user. As a consequence, both the content $g^b, C = g^c, T = e(g, g)^t)$. The adversary must then output a vectors and the attribute vectors of the documents should be taken into consideration in document search process. The simi- An adversary, B, has at least an ε advantage in solving the larity between a pair of documents F_i, F_j with content vectors DBDH problem if

$$Sim(F_i, F_j) = \gamma \cdot RScore(V_i, V_j) + (1-\gamma) \cdot \frac{\quad}{\quad} \quad (7)$$

where $0 \le \gamma 1 \le$ and $RScore(V_i, V_j)$ is the relevance between the content vectors of the two documents and it is calculated as:

$$RScore(V_i, V_j) = V_i \cdot V_j$$

γ is a preset parameter to adjust the importance degrees of the number of non-zero elements in vector . Based on an ARF vector, the centroid of a cluster C can be easily calculated as:

$$c = LS/P$$

and the similarity between cluster C and a document F_j is defined as:

$$Sim(C, F_j) = \gamma \cdot RScore(c, V_j) + (1-\gamma) \cdot \frac{Length(A_{min}}{\quad} \quad (9)$$

where $0 \le \gamma \le 1$ and $RScore(c, V_j)$ is calculated as:

$$RScore(c, V_j) = c \cdot V_j$$

Further, the radius of cluster C is defined as follows:

$$R = \sqrt{\frac{\quad}{P}} \quad 2 \quad \Sigma_{j=1}$$

and it also can be calculated(V−) by/Pthe ARF vector as follows:(10)$_j$

$$R = \sqrt{\quad} \quad (SS - LS^2/P)/P \quad (11)$$

Theorem 1 (ARF Additivity Theorem): If we merge two disjoint clusters with ARF vectors: $ARF_1 = (P_1, LS_1, SS_1, V_{max1}, A_{min1})$, $ARF_2 (P=_2, LS_2, SS_2, V_{max2}, A_{min2})$, the ARF vector of the combined cluster is:

$$ARF = ARF_1 + ARF_2 = (P_1 + P_2, LS_1 + LS_2, SS_1 + SS_2, V_{max}, A_{min})$$

A_{min2}.

Proof: The proof consists of straightforward algebra.

D. DBDH Assumption

Let $\mathbf{G}_0$, $\mathbf{G}_1$ be two groups of prime order p and g is a generator of $\mathbf{G}_0$. The operator e is a bilinear map between $\mathbf{G}_0$ and $\mathbf{G}_1$ as specified in Section IV.B. The challenger chooses at random. Then the challenger flips a fair binary coin v and if $v = 1$, it generates a BDH 5-tuple $(g, A = g^a, B = g^b, C = g^c, T = e(g, g)^{abc})$; otherwise, if $v = 0$,

$$|Pr[B(g, g^a, g^b, g^c, e(g, g)^{abc}) = 1] - Pr[B(g, g^a, g^b, g^c, e(g, g)^t) = 1]| \ge 2\varepsilon \quad (5)$$

where the probability is over the randomly chosen a, b, c, t and the random bits consumed by B. For the convenience of expression, we denote that $P_{BDH}\{(g, g^a, g^b, g^c, e(g, g)^{abc})\}$ and $_{BDRH}(g, g=\{a, g^b, g^c, e(g, g)^t)\}$.

Definition 1: The DBDH assumption holds if no probabilistic polynomial-time adversary has at least ε advantage in solving the above game. (6)

E. Selective-Set Security Game

In this paper, we employ the Selective-Set Security Game [21], [28], [41] to prove our scheme's security. The game is composed of six phases and they are presented as Follows. (7)

Init: The adversary declares an access tree with a set of Attributes S that he wants to be challenged upon.

Setup: The challenger runs the Setup algorithm presented in Section IV to generate the public parameters which are provided to the adversary.

Query Phase 1: The adversary is allowed to issue queries to obtain the secret keys of any access structure with attribute) algorithm.

Challenge: The adversary provides two different messages M_0 and M_1 with equal length to the challenger. The challenger randomly flips a coin 1 }and encrypts M_μ with attribute set S. At last the encrypted message is sent to the Adversary.

Query phase 2: The query phase 1 is repeated.

Guess: Based on the obtained information, the adversary

We say that our scheme is secure if all the polynomial time adversaries have at most a negligible advantage in the game, where the advantage of the adversary is defined as $-|2^1$. Otherwise, we say that the adversary wins the game.

IV. Hierarchical Attribute-Based Document Encryption

A. Monotone Hierarchical Access Tree

Let $A = \{A_1, A_2, \cdots, A_n\}$ be a set of attributes. A collection is monotone: Given $\mathbf{A}$. A monotone access structure of a document is a monotone collection $\mathbf{A}$ comprised of non-empty subsets of A, the sets not in $\mathbf{A}$ are called unauthorized sets. In this paper, we restrict our attention to monotone access structure which is practical considering the characteristics of the problem stated previously.

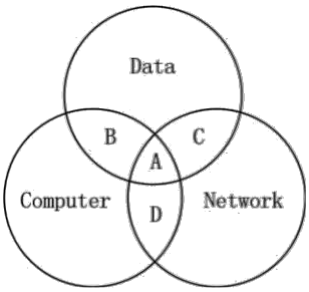

Fig. 2. Assumption of access control mechanism.

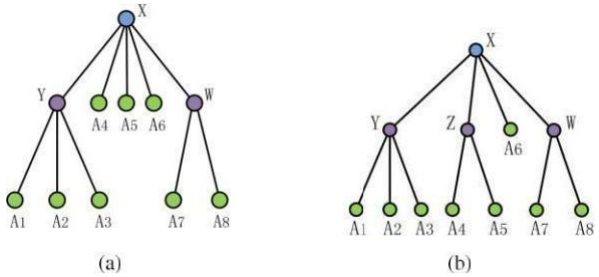

Fig. 3. Examples of access trees.

In this paper, we assume this a file associated with many attributes can be only accessed by the data user who possess all the basic attributes of the file. As an example shown in Fig. 2, the whole data set is divided into three category including computer, network and data related data. Some documents may own two or many attributes such as the documents in region *A, B, C* and *D*. Under our assumption the crossing region *A* can be accessed by the data users who own all the three roles of computer, network researcher and data researcher; region *B* can be accessed by the data users who own the roles of data and computer researcher region *C* can be accessed by the data users who owner the roles of data and network researcher region *D* can be accessed by the data users who own the roles of network and computer researcher. Apparently, under our assumption the access structure of a document is monotone. Take example region *B* a data user who owns the attribute of data and computer researcher can access *B* and then any other data users who have at least these two attributes can also access region *B*.

Let t be a monotone hierarchical access tree represent an integrated access structure for a set of data. The collection of all the access trees is called the access structure of the whole data collection. In this paper each non leaf node of the tree represent a threshold AND gate and associates with a set of attribute which are represented by the leaf nodes. For convenience some function are defined as follows: The number of the child nodes of a non-leaf node x is denoted as *num*. The function *att (x)* denotes the associated attributes with the node *x* and in addition, *att* (F_i) also returns the attribute set associated with document F_i all node in the tree is assign with a numerical identifier and the function *index x* returns the identifier of node *x* In addition *index* (F_i) returns the identifier of fi Note that, each non-leaf node has a unique numerical identifier and the leaf nodes that represent the same attribute in different access tree share a equal numerical identifier. All node in an access tree may contain some files identifiers and the corresponding files will be encrypted by this node.

The function *file(x)* returned the file identifiers contained in node x. as say node y in the access tree t matches a set of attributes s and only if the attribute set of Y equals to s. As shown in Fig. 3a *Y* match s if and only if s = {*A1, A2, A3*} and we denote it as $T_Y(S)$ =0. If here no node in the tree can match *S* we check whether a node

In this tree can cover *S*. We say that node *X* covers *S* if *X* cannot match *S* and the leaf child nodes of *X* composed a superset of s.here denote $T_X(S)$ = 1 if node X covers S. As shown in Fig. 3a node *Y* covers *S* if =*S* {A1, A_2} and node *X* covers *S* if *S* A_4=, {A_5, A_6.}

We constructed the access structure of a data collection in incremental way and an access tree is constructed by continuously splitting the tree in a top down approach. In the initial, we sort the data in decent order based on the number of their attributes. Actually, the attribute set of the first document must be a root node of an access tree and the identifier of the document is inserted to the root node. Given a set of access trees we discuss how insert a new document F_i s identifier into them. The attribute set of the new document *att*(F_i) can be divided into three categories i.e. Being matched by a node in the access trees being covered by a node in the access trees or neither being matched or covered by a node in the access trees. We first has to scan the access trees until finding a node that matches *att*(F_i). If the node exists the identifier of the new document *index* (F_i), is inserted to the node. Otherwise we need to rescan the access trees until find node *X* that can cover *att* (F_i) If the node exists, a new node *Z* is built in the tree to match *att*(F_i) and insert *index*(F_i) into Z . Specifically node *Z* is inserted to the access tree as a child node of *X* and the leaf nodes related with *at* (F_i) is inserted into node *Z*. Meanwhile, we need to delete the leaf nodes from node *X*. As an example, if insert {A_4, A_5 into} the tree presented in Fig. 3a the updated access tree is shown in Fig. 3b. At final if *att*(F_i) neither is matched or covered by a node in the trees we build a new access tree for F_i and insert *index*(F_i) into the root node. The above process is iterated until all the document identifiers are inserted into the access trees. All the access trees composed, the access structure of the whole document collection.

The pseudo-code of incrementally collecting the hierarchical access trees for a document collection is presenting in Algorithm 1. Based on Algorithm 1, a set of integrated access trees are constructed for the document. Note that all the nodes in an access tree compose a monotone access structure and each access tree contains several document identifiers. All the document in a tree can be encrypted together which will be discussed in Section IV.B. The identifier of the node x in an access tree is assigned as follows,

1. If *x* associated with attribute A_i is a leaf node, its numerical identifier is *I*.

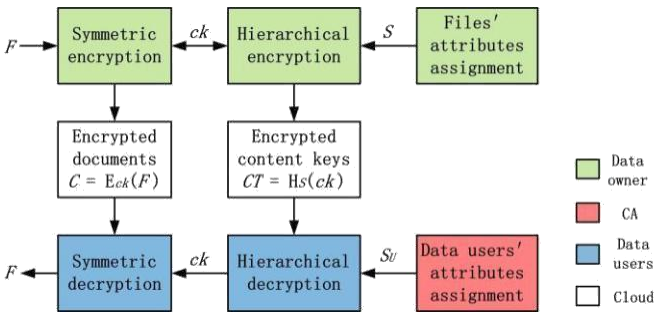

Fig. 4. The flow chart of document encryption and decryption.

Algorithm 1 BuildingAccessStructure

Input: Document collection $F = \{F_1, F_2, \cdots, F_N\}$ with N attribute sets $\{att(F_1), att(F_2), \cdots, att(F_N)\}$
Output: A set of access trees ST

Sort the files in F in descending order based on the number
S=Tnull;
for $i = 1$ N: **do**
  Scan the access trees in order;
  for the scanned access tree T in ST **do**
    if node Y in T matches S, i.e., $T_Y(S) = 0$ **then**
      break;
  else if node X in coversTS , i.e., $x(S)$ 1 **then**T=
    Build a new node Z and let the created node Z be the child of X , and further the leaf nodes associated with S are inserted to Z ; meanwhile, the leaf nodes are deleted from X ;
    break;
  end if
end for
tree **then**
  Build a new access tree for attributes
  Insert the tree to S ; to the root node;
end if
end for

2. If x is a non-leaf node and associated with a set of ordered attributes $\{A_i, A_j, \cdots, A_k\}$ $(i < j < \cdots < k)$, its numerical identifier is $ij \cdots k$.

In this way, each non-leaf node in the access structure has a unique identifier and apparently the leaf nodes associated with a same attribute share a same identifier.

B. Hierarchical Document Encryption

We first describe the system model of hierarchical attribute-based document encryption scheme as shown in Fig. 4. The data owner first selects a set of content keys $ck = \{ck_1, ck_2, \cdots, ck_N\}$ which are used to encrypt the documents in F Symmetrically. Then, the content keys are hierarchically encrypted by the attributes assigned by the data owner. The encrypted documents, access structure and encrypted matter keys are outsourced to the cloud server. In addition, the index structure of the document collection is also stored in the cloud server to support document search and it will be discussed in Section V. Once the encrypted search results are sent to the data users, they decrypt the content keys by their secret keys and further decrypt the documents based on the decrypted content keys. In the following, we mainly discuss how to encrypt the content keys in detail.

We first introduce the conceptions of bilinear map and Lagrange interpolation which are involved in our scheme. Let $\mathbf{G}_0$ and $\mathbf{G}_1$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbf{G}$ and e be a bilinear map, $e: \mathbf{G}_0 \times \mathbf{G}_0 \rightarrow \mathbf{G}_1$ with the following properties:

1. Non-degeneracy: $e(g, g) \neq 1$.
2. $e(u^a, v^b) = e(u, v)^{ab}$.

In addition, $\mathbf{G}_0$ is a bilinear group if the group operations in $\mathbf{G}_0$ and the bilinear map $e: \mathbf{G}_0 \times \mathbf{G}_0 \rightarrow \mathbf{G}_1$ are both efficiently computable. The Lagrange Coefficient $\Delta_{i,S}$ for $i \in \mathbb{Z}_p$ and a set, S, of elements in $\mathbb{Z}_p$ is defined as: $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x - j}{i - j}$. In addition, a hash function $H: \{0,1\}^* \rightarrow \mathbf{G}_0$ is employed to map the string attributes to a random group element in $\mathbf{G}_0$.

The detailed process of encrypting the documents is presented in the following:

1) Setup: Each document in F is assigned with a set of attributes and the access structure of the document collection is constructed based on Algorithm 1. A set of content keys $ck = \{ck_1, ck_2, \cdots, ck_N\}$ are randomly selected for the files in F which are used to encrypt the files symmetrically. Then the setup algorithm chooses a bilinear group $\mathbf{G}_0$ with g as a generator, a bilinear map $e: \mathbf{G}_0 \times \mathbf{G}_0 \rightarrow \mathbf{G}_1$ and two random numbers $\alpha, \beta \in \mathbf{Z}_p$. The public key is published as:

$$PK = (\mathbf{G}_0, g, h = g^{\beta}, e(g, g)^{\alpha})$$

and the master secret key MSK is (β, g^{α}).

2) Encrypt(PK, ck, ST): For each attribute A_i in A, we first randomly select a unique secret number $s_i \in \mathbb{Z}_p$. x in the access trees. In each access tree, these secret numbers for the nodes are chosen in a bottom-up manner, starting from the leaf nodes to the root node. The number sk_x of the leaf node x associated with attribute A_i is set as s_i . Then for the non-leaf node x with a set of child nodes S_x, the secret number sk_x is computed as $sk_x = \sum_{i \in S_x} sk_i \cdot \Delta_{i,S_x}(index(x))$ where $S_x = \{$ of node x . By iterating the above process, each node in the access structure can be assigned with a secret number. numerical identifier$_x$

Then, the content keys are encrypted by the secret numbers of the nodes in the access trees. As presented in Algorithm 1, each node x contains a set of file identifiers $\{f_m, \cdots, f\}_n$ which can be returned by $file(x)$. We encrypt all the

corresponding content keys $\{ck_m, \cdots, ck_n\}$ by the same secret number sk_x. Specifically, for each access tree T in ST, let Y be the set of leaf nodes in T. All content keys related with are T encrypted together and the ciphertext is constructed as follows:

that, several leaf nodes $y_1, y_2, \cdots, y_d$ of different access trees $T_1, T_2, \cdots, T_d$ may share a same attribute A_i and in this case, $= H(A_i)^s$. Therefore, in the ciphertext of the whole

2) Decrypt(CT, SK): We employ a recursive algorithm DecryptNode(*CT, SK, x*) to decrypt the content keys. This algorithm takes a ciphertext *CT*, a private key *SK* which is associated with a set of attributes *S*, and a node *x* from T as input.

S, the algorithm is defined as follows:

$$= \frac{e(D_i, C_x)}{e(g^r \cdot H(A_i)^{ri}, h^{sk_x})} = \frac{e(g^r, h^{sk_x}) e(H(A_i)^r, h^{sk_x})}{e(h^{r_i}, H(A_i)^{sk_x})} = e(g, g)^{r\beta \cdot sk_x}$$

When x is a non-leaf node, the algorithm is operated recursively. Specifically, it processes as follows: we denote the set of x's children nodes by S_x. For each node S_x, it calls DecryptNode*(CT, SK, z)* and stores the output as F_z. returns Otherwise, $SK, x) = x$

$$F_x = \prod_{z} F_z = e(g, g)^{r\beta \cdot sk_x}$$

If a data user with a set of attributes $\beta \cdot k$ S that satisfies the sub-tree T_x, the data node x with sk_x can be decrypted by computing $C_i / (e(g^{\alpha} \cdot h^r)/e(g, g)^{r\beta \cdot sk_x}) = ck_i$. At last, all the documents encrypted by ck_i can be decrypted. Otherwise, the data user cannot decrypt the documents.

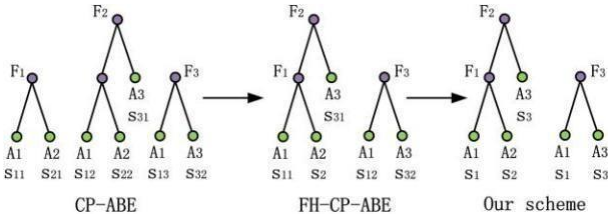

Fig. 5. Comparison of CP-ABE, FH-CP-ABE and our scheme.

Note that, in the encryption phase, the secret numbers of the nodes are chosen in a bottom-up manner which is totally different from existing schemes such as CP-ABE and FH-CP-ABE. An advantage of this manner is that all the same attributes in different access trees share a same secret number and this can significantly improve the flexibility of encryption, decryption and secret keys distribution. As an example, shown in Fig. 5, three files F_1, F_2, F_3 are associated with attribute sets $\{A_1, A_2\}$, $\{A_1, A_2, A_3\}$ and $\{A_1, A_3\}$, respectively. In CP-ABE, the three files are encrypted individually and attribute A_1 is related with three random secret number s_{11}, s_{12}, s_{13}, A_2 is related with s_{21}, s_{22}, A_3 is related with random secret number s_{31}, s_{32}. In FH-CP-ABE, file F_1, F_2 share an access structure and they are encrypted jointly. File F_3 is encrypted individually. In this case, attribute A_1 is related with two secret number s_{11}, s_{12}. Attribute A_2 is related with s_2 and attribute A_3 is related with s_{31}, s_{32}. In our scheme, each attribute is related with only one secret number.

V. Efficient Retrieval Over Encrypted Document Collection

In this section, an efficient retrieval scheme over encrypted document collection is designed and we first describe the process of constructing the ARF tree. Then a depth-first searching algorithm of the ARF tree is designed and in addition, it can be operated in a parallel manner flexibly. Given a collection of documents $F = \{F_1, F_2, \cdots, F_N\}$, each document needs to be scanned for one time and the number of each keyword is recorded. Then a normalized vector for the document is created based on the keyword word list. W as discussed in Section III.B. The attribute vector of a document can be built based on attribute dictionary A and the associated attributes appointed by the data owner. Organizing the document vectors properly can radically improve the search efficiency. In some encrypted document recovery schemes [17], [18], the document content vectors are ordered randomly, and the search difficulty is $O(N)$, where N is the number of documents. To improve search efficiency, in some other schemes [15], [16], the vectors are arranged based on their relative comparisons and they can obtain sub- linear search efficiency. However, the search accuracy cannot be sure. In our scheme, the similarity between a pair of documents is calculated based on both the content vectors and attribute vectors. The planned scheme can always obtain the accurate search results with at least a sub-linear search productivity.

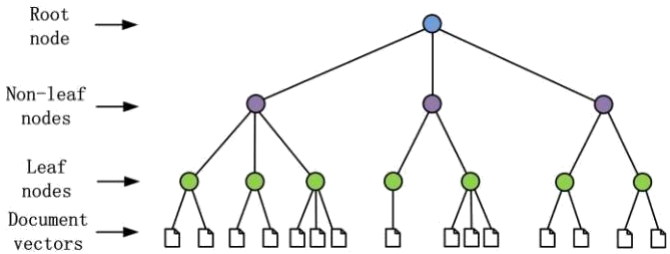

Fig. 6. An ARF tree.

For convenience sake, we first describe the structure of an ARF tree briefly. An ARF tree is presented in Fig. 6 and it can be observed that the ARF tree is a height-balanced multi-way tree. An ARF tree has three main parameters including branching factors K_1, K_2 and threshold T which are preset by the data owner. A leaf node L_i contains at most K_1 document vectors and it is defined as follows:

$$L_i = (ARF, child_1, \cdots , child_j), 1 \le j \le K_1$$

where ARF is the ARF vector of the cluster, $child_j$ is a cursor to the j -th document vector in the cluster. Each leaf node represents a micro cluster collected of a set of document vectors. Each non-leaf node NL_i contains at most K_2 child nodes and it is defined as follows:

$$NL_i = (ARF, ARF_1, child_1, \cdots , ARF_j , child_j), 1 \le j \le K_2$$

where ARF is the ARF vector of the whole gathering characterized by NL_i , ARF_j is the ARF vector of the j -th sub-cluster and $child_j$ is a pointer to the child node representing the sub- cluster. Therefore, a non-leaf node represents a cluster made up of all the sub-clusters represented by its child nodes. Further, the cluster of a leaf node must satisfy a threshold requirement: the radius of the cluster which can be calculated by (11) has to be less than T .

We construct the ARF tree in an incremental manner which is similar to the construction process of the CF tree [42]. The process of inserting a document F_j with vector $\{V_j , V_j^+\}$ into the ARF tree is presented as follows:

- *Identifying the appropriate leaf node:* Starting from the root, F_j recursively descents the ARF tree by choosing the most similar child node according to the similarity scores between F_j and the sub-clusters as defined in (8) until it reaches a leaf node.
- *Modifying the leaf node:* When F_j reaches a leaf node L_i . If so, $Vj\{ , Vj$ is* $\}$ inserted into Li and the ARF vector of Li is updated based on Theorem 1 as discussed in Section III.C. If not, we must split Li to two leaf nodes. Node splitting is done by choosing the farthest pair of document vectors based on (5) as seeds, and then redistributing the remaining document vectors based on the closest criteria. The ARF vectors of the two new leaf nodes need to be recalculated.
- *Modifying the path from the root node to the leaf node.* After inserting $\{ V_j , V_j^+ \}$ into a leaf node, we need to update the ARF vector for all the nodes on the path to the leaf node L_i .

In the absence of a split, this simply involves updating ARF vectors based on Theorem 1. A leaf node split involves us to insert a new leaf node to the parent node. If the parental node has space for the new leaf node, we just need to insertion the new leaf node into it and then update ARF vector for the parent node. In general, however, we may have to split the parent node as well, and so up to the root. If the root is divided, the tree height increases by one.

Except for K_1, K_2, and T , the parameter γ can also affect the structure of the ARF tree. If γ is set to 1, the documents will be arranged based on their matter only and the attendant attributes are unnoticed. On the contrary, if we set γ as 0, the attributes of the documents decide the ARF tree and the content of the documents are not employed. In general, we can set γ as a number between 0 and 1 to balance the important degrees of documents' contents and attributes.

Another challenge is searching the top-k relevant documents whose attributes are covered by the data users. We design a depth-first search algorithm for the ARF tree and the pseudo-code is presented in Algorithm 2. For convenience, some symbols and functions are first defined as follows:

- *kth Score* - The smallest relevance score in current result list *RList* which stores the most k relevant legal accessed document vectors with V_Q and the corresponding relevance scores in order.
- *RScore*(u, V_Q) - The relevance score between the cluster represented by node u and a query vector V_Q is defined as *RScore*$(u, V_Q) = \cdot c\ V_Q$ where c is the center of the cluster.
- *Stack* - We employ the variable *Stack* to store the nodes which need to be searched in the future. In addition, *Stack*.push(u) inserts node u into *Stack* and *Stack*.pop() returns the latest inserted node.

V) = *Length*(*V*).

), V_1 is covered by V_2 ; otherwise, V_1 is not covered by V_2

As shown in line 1 to line 9 in Algorithm 2, we first need to initialize *RList* by finding the most similar leaf node. Then, as shown in line 10 to line 22, the paths in the tree needed to be searched are selected by criteria *RScore*$(V_{u,max}, V_Q) >$ *kthScor e* and *Length*$(A_{u,min}$ *Length*$(A_{u,min})$. This is reasonable considering that if *RScore*$(V_{u,max}, V_Q)$ ≤ *kthScor e* for a cluster, it is impossible that any document vector in the cluster can be a candidate of the search result because the elements in V_Q and V_{max} are naturally nonnega-tive. In addition, if *Length*$(A_{u,min}$

for a cluster, all the attributes of the documents in the cluster cannot match that of the data user. Therefore, this cluster is also unnecessary to be searched. However, if a leaf node is searched, the result list *RList* needs to be updated. In this way, quite many paths are pruned and hence the search efficiency greatly improves. Once the top-k relevant documents are located in the ARF tree, the subsequent encrypted documents are sent to the data user. Apparently, these permissible documents can be decrypted by the data user and then the document query process is finished.

Algorithm 2 DepthFirstSearch

Input: an ARF tree with root r, a query vector V_Q, an attribute vector $V_{Q'}$ of the data user

Output: The most k relevant legal document vectors

1: $u \leftarrow r$;
2: **while** u is not a leaf node **do**
3: **for** all the child nodes v of node u **do**
4: Calculate the relevance scores between v with V_Q by $RScore(v, V_Q)$;
5: Check whether the attribute set $A_{v,min}$ by comparing $Length(A_{v,min}$ $A_{v,min})$;
7: **end for**
8: **end while**
9: Select the most relevant k document vectors in the leaf node u whose attributes are covered by $V_{Q'}$ and construct $RList$;
10: $Stack$.push(r);
11: **while** $Stack$ is not empty **do**
12: $u \leftarrow Stack$.pop();
13: **if** the node u is not a leaf node **then**
14: **if** $RScore(V_{u,max}, V_Q) > kth\ Score$ and $Length(A_{u,min}$
15: Sort the child nodes of u in ascent order based on the relevant scores with VQ whose attribute sets are covered by VQ';
16: Push the children of u into $Stack$ in order, i.e., the most relevant child is latest inserted into $Stack$;
17: **end if**
18: **else**
19: Calculate the relevance scores between the document vectors in the leaf node with V_Q and compare their
20: Update $RList$;
21: **end if**
22: **end while**
23: **return** $RList$

We can further improve the search efficiency by operating the searching process in parallel. In the search process, all the processors need to share the same result list $RList$. Assume that there is a set of processors $P = \{p_1, p_2, \cdots, p_l\}$ and given a search request, an idle processor p_i is used to find most relevant leaf node on the tree and initialize $RList$. Then, all the necessary search paths are selected based on

criteria $V_Q \cdot V_{max} > kthScore$ and $Length(A_{min} = Length(A_{min})$. If the search process can be continued on q search paths and there are more than q idle processors, any q processors are selected and each processor is responsible for searching a child path. If there is q^* ($q^* < q$) idle processors, they search the latest inserted q^* children paths in $Stack$. Once an idle processor appears, it continue to search the node generated by $Stack$.pop(). At last, the most relevant k encrypted files (i.e., the search result SR) are sent to the data user and they are decrypted by the secret key of the data user.

Though the document retrieval efficiency is greatly better based on the ARF tree, a companying challenge is how to protect the privacy of the document vectors in the index structure and query vectors. Fortunately, this problem has been widely discussed and researched [15]–[17], [37]. In this paper, we strictly employ the method in [18] to protect the security of the document vectors while preserving the searchability.

VI. Security Analysis

In the document retrieval system, the cloud server and CA center are assumed to be trustable. In this section, we focus on the security of the proposed hierarchical document encryption scheme and its security mostly includes two aspects including document privacy and content keys confidentiality. The documents are encrypted based on symmetric encryption.

schemes (e.g., AES) with content keys and their security is out of the scope in this paper. In this section, we analyze the security of the content keys which are encrypted by the proposed hierarchical encryption scheme. We provide the Decisional Bilinear Diffie-Hellman [28], [41], [43] assumption (DBDH) in Section III.D and Selective-Set Security Game is given Section III.E. In this section, we reduce the security of the content keys to the hardness of the DBDH and prove the security of the proposed scheme under the Selective- Set Security Game.

Theorem 2: Under the DBDH assumption, no polynomial time adversary can win the Selective-Set Security Game.

Proof: Suppose there exists an polynomial adversary Adv that can break our scheme with an advantage ε. We can design a simulator B that can play the DBDH game with an advantage ε_2. The game is executed as follows:

First, the challenger chooses G_0, G_1, g, a, b, c, t and a bilinear map e as specified in Section IV.B. Then he randomly flips a fair binary coin v and if $v=1$, $T = e(g,g)^{abc}$, i.e., $(g, A = g^a, B = g^b, C = g^c, T = e(g,g)^{abc}) = R_{BDH}$; otherwise, if $v = 0$, $T = e(g,g)^t$, i.e., $(g, A = g^a, B = g^b, C = g^c, T = e(g,g)^t) = R_{BDH}$. The challenger sends $(g, A, B, C, T) = (g, g^a, g^b, g^c, T)$ to the simulator B. The simulator B now plays the role of challenger in the security game. Then, the security game are executed as follows:

Init: The adversary Adv submits the simulator B a set of attributes S that it wants to be challenged upon.

Setup: The simulator B sets α is randomly selected from Z_p and it computes $e(g,g)^{\alpha + ab} = e(g,g)^{\alpha} e(g,g)^{ab}$. It further sets $h = g^{\beta} = g^b = B$. At last, the public key PK is sent to the adversary Adv. The simulator B randomly chooses $r_j \in Z_p$ and calculates $D_j = g^{r_j}$ and $D'_j = B^{r_j H(A_j)}$. At last, the secret key SK is sent to the adversary Adv.

TABLE I
COMPARISON OF CP-ABE, FH-CP-ABE AND OUR SCHEME

Component	CP-ABE	FH-CP-ABE	Our Scheme
Encryption Time	$[2(\|\mathbb{A}_{C_1}\|+\cdots+\|\mathbb{A}_{C_N}\|)+N]C_{\mathbb{G}_0}+NC_{\mathbb{G}_1}+NC_e$	$Null$	$(2\|\mathcal{A}\|+\rho N)C_{\mathbb{G}_0}+NC_{\mathbb{G}_1}+NC_e$
Decryption Time	$2[(\|t_1\|+\cdots+\|t_N\|)+N]C_{\mathbb{G}_1}+[2(\|\mathbb{A}_{C_1}\|+\cdots+\|\mathbb{A}_{C_N}\|)+N]C_e$	$Null$	$2N(\rho+1)C_{\mathbb{G}_1}+(2\|\mathcal{A}\|+N)C_e$
The Size of PK	$3L_{\mathbb{G}_0}+L_{\mathbb{G}_1}$	$3L_{\mathbb{G}_0}+L_{\mathbb{G}_1}$	$2L_{\mathbb{G}_0}+L_{\mathbb{G}_1}$
The Size of MSK	$L_{\mathbb{Z}_p}+L_{\mathbb{G}_0}$	$L_{\mathbb{Z}_p}+L_{\mathbb{G}_0}$	$L_{\mathbb{Z}_p}+L_{\mathbb{G}_0}$
The Size of SK	$[2(\|\mathbb{A}_{C_1}\|+\cdots+\|\mathbb{A}_{C_N}\|)+N]L_{\mathbb{G}_0}$	$[2(\|\mathbb{A}_{C_1}\|+\cdots+\|\mathbb{A}_{C_N}\|)+N]L_{\mathbb{G}_0}$	$(2\|\mathcal{A}\|+1)L_{\mathbb{G}_0}$
The Size of CT	$[2(\|\mathbb{A}_{C_1}\|+\cdots+\|\mathbb{A}_{C_N}\|)+N]L_{\mathbb{G}_0}+NL_{\mathbb{G}_1}$	$Null$	$(2\|\mathcal{A}\|+\rho N)L_{\mathbb{G}_0}+NL_{\mathbb{G}_1}$

Challenge: For convenience's sake, we assume that only one file is encrypted and consequently the ciphertext can be simplified as $C=\tilde{e}^{\alpha}M_{\mu}\cdot T\cdot e(g,g)^{s}, C_y=g^{s}\cdot H(att(y))^{s}$). The adversary *Adv* submits two messages M_0 and M_1 with equal lengths to *B*. The simulator *B* randomly flips a coin $\mu\in$ {0, 1} and encrypts M_μ with attribute set $C=g^{c}=C$. Suppose that the simulator is given a BDH tuple, that is $T=e(g,g)^{abc}$. Then we have $C=\tilde{e}^{\alpha}M_{\mu}\cdot e(g,g)^{abc}=M_{\mu}\cdot e(g,g)^{\alpha}=M_{\mu}\cdot T\cdot e(g,g)^{c}$. We see that the ciphertext is a valid encryption of M_μ. Otherwise, we have that $T=e(g,g)^{t}$ is a random element of $\mathbf{G}_1$. In that case the ciphertext will give no information about the simulator's choice of μ. At last, the *CT* is sent to *dAv*.

Query phase 2: The query phase 1 is repeated.

Guess: The adversary *Adv* makes a guess μ' of μ based on the obtained information. At the same time, the simulator *B* also makes the corresponding guess of *v* in playing the DBDH game based on the different results the adversary *Adv* guessed. If $\mu'=\mu$, *B* outputs guess $v=1$ in playing the DBDH game and points out that the challenger given 5-tuple to it which is selected from R_{BDH} in playing the DBDH game. If $\mu'\neq\mu$, *B* outputs guess $v=0$ in playing the DBDH game and points out that the challenger given 5-tuple to it which is selected from R_{BDH}.

The probability that the simulator *B* successes in playing the DBDH game between simulator and challenger is calculated as follows.

If $v_b=1$, the challenger generates a BDH tuple $(g, g^{a}, g^{b}, g^{c}, e(g,g)^{abc})$. Then we have $|\mathbf{A}_{C_1}|+\cdots+|\mathbf{A}_{C_N}|)$ and $\rho N<N|t_1|+\cdots+|t_N|$.

we see that *CT* is a valid encryption of M_μ and by definition, in this case the adversary *Adv* has a non-negligible advantage ε calculated as $\Pr[\mu'=\mu|(g,g^{a},g^{b},g^{c},e(g,g)^{abc},v=1)]=\frac{1}{2}+\varepsilon$.

If $v_b=0$, the challenger builds a random 5-tuple $(g, g^{a}, g^{b}, g^{c}, e(g,g)^{t})$, i.e. $(g, A, B, C, T)\in R_{BDH}$. Then we have that *T* is a random element of $\mathbf{G}_1$. The adversary *Adv* did not get any information about the message M_μ, so there is no advantage to guess the correct μ'. As a consequence, the adversary can make a correct choice with a probability $\frac{1}{2}$. Therefore, the probability of a correct choice is $\Pr[\mu'=\mu|(g,A,B,C,T)\notin R]=\frac{1}{2}$.

At last, the overall advantage of *B* in playing the DBDH game can be calculated as $\Pr[\mu'=\mu|(g,A,B,C,T)\in R_{BDH}]-\frac{1}{2}=\frac{1}{2}+\varepsilon-\frac{1}{2}=\varepsilon$.

Based on the definition of DBDH assumption, we can infer that our scheme is secure. The theorem is proved.

VII. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the hierarchical document encryption scheme and in addition the search efficiency of the ARF tree. We first analyze the efficiency the-Operatically and then conduct experiments to verify the analysis result.

A. Theoretical Analysis

We compare our hierarchical encryption scheme with CP-ABE scheme in [21] and FH-CP-ABE scheme [28], and some definitions are defined first. We assume that $C_{\mathbb{G}_i}$ $(i=0,1)$ is the time cost of an operation on the group such as exponentiation or multiplication. Let $\mathbf{Z}_p$ be the group $\{0, 1, \cdots p-1\}$ and C_e be the cost of an bilinear map operation *e*. Let *N* be the number of documents in the collection, ρ be a parameter that associated with Algorithm 1 and ρN is the number of the nodes in all the access trees. Considering that a set of file identifiers share a node in the access trees, ρ is naturally smaller then 1. Let A, $\mathbf{A}_u$, $\mathbf{A}_{Ci}$ be the attribute dictionary, the attributes associated with the data user and document *F* respectively. Let t_i be the number of interior nodes in the access tree of file F_i. In addition, we define $|t_i|$ as the number of elements in t_i.

In the analysis, we assume that the data owner has *N* document files and their content keys are encrypted by CP-ABE, FH-CP-ABE and our scheme. Note that, we focus on the encryption process of the content keys rather than that of the documents which are encrypted by the content keys symmetrically. We further assume that a data user is responsible for decrypting all the documents and the analysis result is presented in Table I. For a large document set, we have $|A|$ and $\rho N<N|t_1|+\cdots+|t_N|$.

As a consequence, our scheme performs better then CP-ABE in time costs of encryption and decryption, and the sizes of *PK*, *SK* and *CT*. The two schemes have same performance in the size of *MSK*. In conclusion, our scheme can improve time and storage efficiency compared with CP-ABE. For a constant attribute set *A* and parameter ρ, the encryption time, decryption time and size of *CT* all increase linearly with the number of documents in our scheme. The sizes of the keys are independent of the document collection. In addition, our scheme outperforms FH-CP-ABE in terms of the size of *PK* and *SK* and they have similar performance in terms of *MSK*. Considering that FH-CP-ABE is designed to encrypt a set of documents with incremental attribute sets, i.e., $\mathbf{A}_{C1}\supseteq\mathbf{A}_{C2}\supseteq$

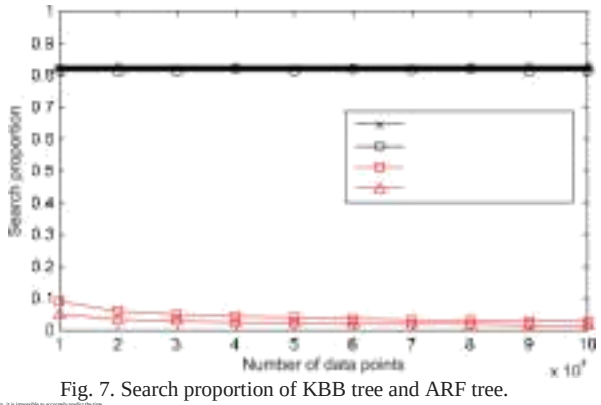

Fig. 7. Search proportion of KBB tree and ARF tree.

cost of encryption and decryption and the size of *CT* for a document collection with randomly assigned attribute sets. As a consequence, we will further compare our scheme with FH-CP-ABE by simulation in Section VII.B.

The association structure of the document collection affects the search proficiency significantly. The keyword balanced binary (KBB) tree can provide accurate search result. However, the document vectors are randomly inserted into the tree and they are organized chaotically. Some similar vectors may locate very far in the tree and some totally different vectors may be neighbors with each other. Therefore, the interior nodes in the tree can provide very limited knowledge to lead a query vector to the area with a set of strongly relevant document vectors. On the contrary, the vectors in the ARF tree are formed strictly giving to their similarities and similar vector can always compose a group in spite of the vectors' input order. The query vector can easily locate a cluster that contains relevant document vectors. The search proportion is defined as the proportion that the document vectors being searched in a search process and it is computed by the number of the searched nodes to the number of all the nodes in the tree. A basic comparison between the two trees is presented in Fig. 7. All the document vectors are randomly created in 2D and 3D space. To be fair, we ignore the attributes of data user and documents considering that the KBB tree does not support attribute limited search. It can be observed that the ARF tree outperforms KBB tree significantly in both 2D and 3D spaces. Specifically, the search proportion of ARF tree is about 5% to 10% to that of KBB tree.

B. Experimental Simulation

We organize a thorough experimental evaluation for the proposed document recovery scheme on a real-world data set: the Enron Email Data Set. The data set is first treated, and 1,000 records are casually chosen as our testing corpus. We implement the hierarchical encryption scheme based on CP-ABE toolkit and Java Pairing-Based Cryptography library. The document search process is implemented based on Java language. All the following experiments are conducted on a 2.6 GHZ Intel Core i5 processor, Windows 7 operating system with a RAM of 4G.

1) Effectiveness of the Integrated Access Trees: The attribute set is defined as $A = \{A, B, \cdots, Z\}$ which

Algorithm 3 AttributeGeneration

Input: $A = \{C_1, C_2, C_3, C_4\}, F, p_r$ $(0.25 \leq p_r \leq 1)$
Output: The attributes of each document
3: Randomly select a number m from $1, \{2, 3, 4, 5;\}$
5: Insert A_n to Att;
6: **for** $i = 2 : m$ **do**
8: Insert A_q to Att;
9: **end for**
10: The attributes in Att is defined as the attributes of document F_i;
11: **end for**

is composed of 26 letters. Then, all the attributes are divided into 4 categories, i.e., C_1 *A, B, G, and=C2*{ ⋯} = { ⋯} ={ ⋯ } ={ ⋯ } The related attribute of a document is randomly generated through Algorithm 3. We assume that each document has at least 1 attribute and at most 5 attributes. As shown in line 5 of Algorithm 3, the attributes of a document trend to belong to one attribute category with a larger probability p_r. This is agreeable considering that the attributes are associated with each other and if a set of attributes are strongly related, they are likely to belong to a document jointly. For example, if a document is related with "computer", it is more likely to be also related with "network" rather than other attributes such as "economic" and "finance".

Parameter p_r affects the access trees greatly as offered in Fig. 8. For a constant p_r, the number of the access trees monotonously increases with the number of files as shown in Fig. 8(a). When p_r is set to 1.0, all the attributes of a file fall in a sub-category of and in this *A*case the number of access trees is the lowest. Note that, a small number of access trees can lead a high encryption and decryption efficiency, because many documents share an access tree and they can be encrypted together in this case. When we decrease p_r from 1.0 to 0.3, the attributes of a file are other and more likely to be picked from the whole attribute set*A* at random and the diversity of the documents' attributes increases. Consequently, the number of the access trees increases. In the worst case, i.e., p_r is set to 0.25 and the attributes of a file are wholly randomly selected from ,*A*the number of access trees is the largest with a constant number of files. In CP-ABE, each document has an access tree and the number of all the access trees totals to the number of files which is much larger than that of the planned scheme. As shown in Fig. 8(b), the number of nodes in the access trees has similar pattern with the number of access trees and the planned scheme always plays better than CP- ABE.

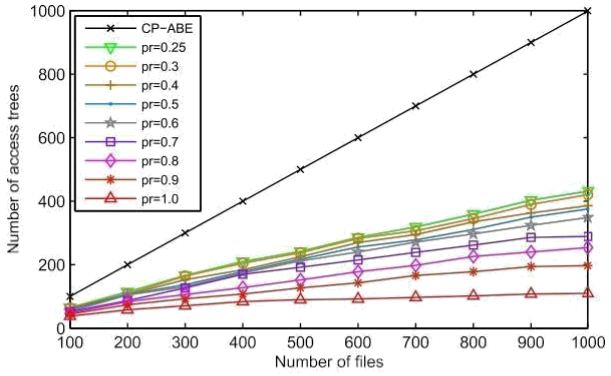

Fig. 8. Number of access trees and that of nodes in the trees with different p_r and number of files.

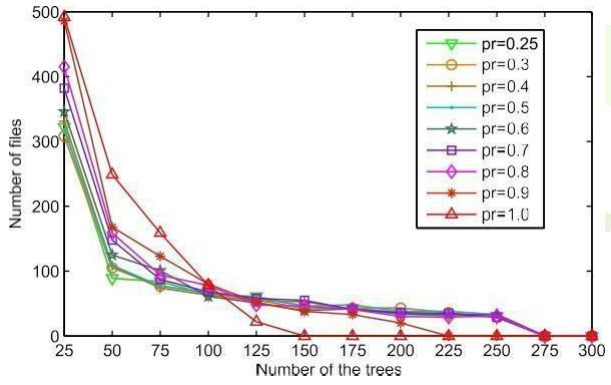

Fig. 9. Distribution of files in the access trees.

We further analyze the sharing of files in the access trees and reproduction result with N 1, 000 is provided in Fig. 9. The access trees are first falling sorted according to their sizes, i.e., the number of nodes in the trees, and then the numbers of files in the trees are calculated. It can be observed that about 30% to 50% files are covered by the 25 largest trees and about 40% to 80% files are contained by the 50 largest trees. In addition, the files trend to aggregate with each other to some more trees with the increasing of When we set p_r to 1, more than 90% files are covered by the largest 100 access trees and, most of the other trees contain a small number of nodes and they may cover 1 or 2 files. Without loss of generalization, in the following, we think that p_r equals to 0.9.

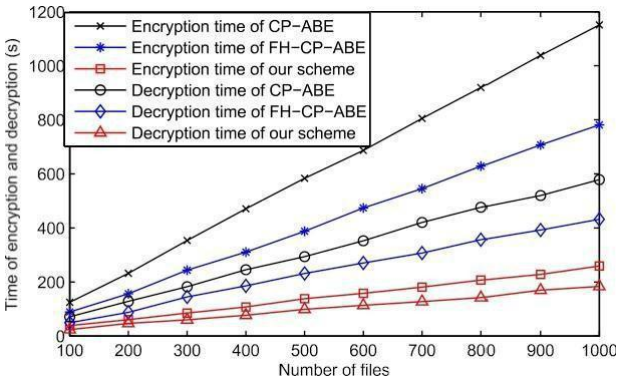

Fig. 10. Efficiency of encryption and decryption.

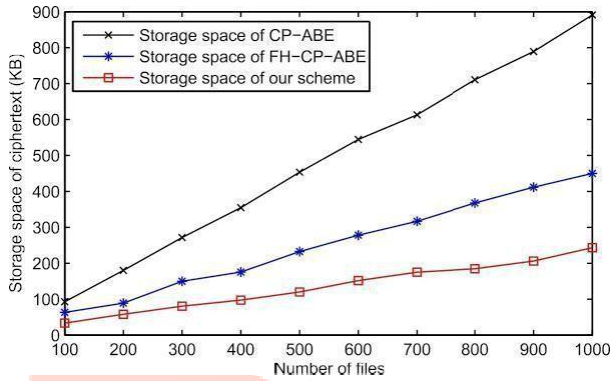

Fig. 11. Storage space of the cipher text *CT*.

2) Efficiency of Hierarchical Document Encryption: The time consumptions of encrypting and decrypting the whole document collection are offered in Fig. 10. In CP-ABE, each document is encrypted and decrypted independently. Consequently, the time of both encryption and decryption rises almost linearly with the number of files. On the contrary, a set of files in our scheme share an access tree and they are encrypted and decrypted together. The encryption and decryption time increases logarithmically with the number of files. Apparently, the planned scheme is much more time efficient than CP-ABE. Though the FH-CP-ABE performs a little better than CP-ABE, it cannot efficiently encrypt and decrypt a document collection considering that the number of unified access trees are much larger than that of our system. The storage space of the cipher text is presented in Fig. 11. Note that, only the encrypted content keys are considered in this experiment and the symmetrically encrypted documents are not believed. The storage space of CP-ABE linearly increases with the number of files and it can be clarified by the fact that each file has a content secret key which is encrypted separately. In our scheme, if a set of files have similar attribute sets, they may share an access structure and their content keys are related with each other. In addition, a set of files can share a same content key if they have the same attribute sets. Consequently, the planned scheme is more space-efficient than CP-ABE. Similar to the efficiency of encryption and decryption, FH-CP-ABE performs better than CP-ABE and worse than our scheme.

3) Efficiency of Document Retrieval: Except for providing an efficient document encryption scheme, we also improve

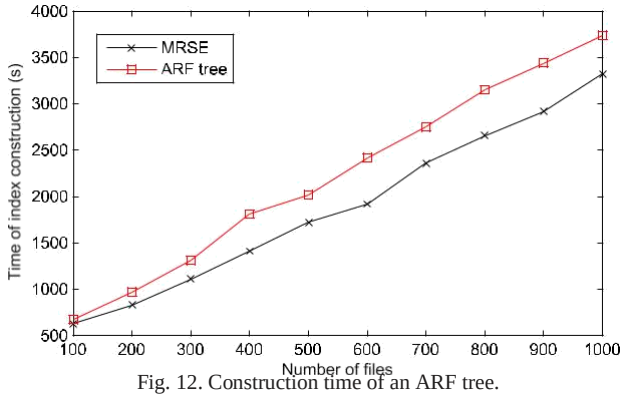

Fig. 12. Construction time of an ARF tree.

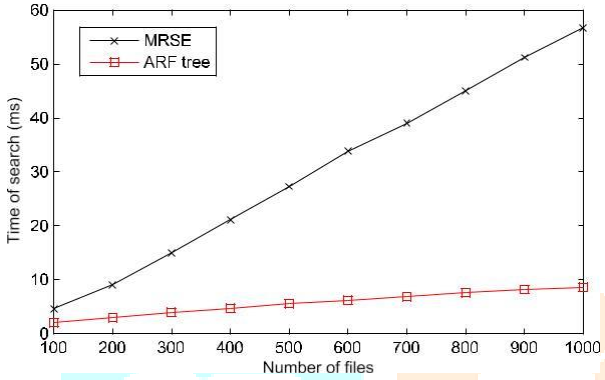

Fig. 13. Search time of a query.

The search productivity matched with MRSE. Note that, in our replication, the index structures of both MRSE and ARF are plaintext. The production time of an ARF tree is strongly related with the number of files and it is offered in Fig. 12. The index construction times of both the two schemes linearly growth with the number of files. This can be clarified by the fact that most time is expended in the process of generating document vectors (about 3.2 seconds/file). The ARF tree spends slightly more time than MRSE, because the document courses need to be inserted into the tree.

Another dimension of our scheme is the search productivity. In the Enron Email Data Set, the documents have no attribute which should be appointed by the data owner. In general, the attributes of the documents are related with their con- shelters. Though, in Algorithm 3, the attributes of a document are casually selected, and they may misinform the ARF tree structure process. Therefore, for accessibility, in the following we set γ equals to 1 when building the ARF tree. In addition, k is set as 10 (i.e., 10 encrypted documents are returned for a query). However, the attributes are employed in the document search process and the replication result is provided in Fig. 13. Apparently, the search time in MRSE linearly increases with the number of files considering that the document routes are arranged randomly, and all the document paths need to be checked for one time. However, the ARF tree arranges the files based their resemblances which greatly improve the search efficiency. In Particular, quite a number of the search paths are cut in the search process and ARF tree has logarithmic time utilization with the No of files.

VIII. CONCLUSION

In this paper, we consider a new encrypted document retrieval scenario in which the data owner wants to monitor the documents in fine-grained level. To help this service, we first design a novel classified attribute-based document encryption scheme to encrypt a set of documents jointly that share an integrated access structure. Further, the ARF tree is proposed to organize the document vectors based on their parallels. At last, a depth-first search algorithm is designed to improve the search efficiency for the data users which is extremely important for large document collections. The performance of the approach is completely calculated by both abstract analysis and experiments.

The suggested scheme can be further increased in several aspects: First, in this paper, we assume that each node in the access trees represent an “AND” gate and this limits the springiness of assigning the attributes to the documents. In the future, we will attempt to introduce “OR” gates into the access trees. Second, the access structure of the document collection is generated in a greedy manner and we will check whether it can be further improved to reduce the number of access trees. In addition, the withdrawal method of the data users’ attributes needs to be designed. Third, the update strategy of the ARF tree should be proposed. Though the ARF tree naturally supports adding new nodes to the tree, the method of erasing a node from the tree did not provided. Fourth Part, a new document collection, in which each file is associated with a set of proper characteristics, should be developed and a methodical experiment should be conducted on the collection to test the love of issue γ on the approach.

REFERENCES

[1] K. Ren, C. Wang, and Q. Wang, “Security challenges for the public cloud,” *IEEE Internet Comput.*, vol. 16, no. 1, pp. 69–73, Jan. 2012.

[2] D. X. Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in *Proc. IEEE Symp. Secur. Privacy*, 2000, pp. 44–55.

[3] E.-J. Goh, “Secure indexes,” Cryptol. ePrint Arch., San Francisco, CA, USA, Tech. Rep. 216, 2003. [Online]. Available: http://eprint. iacr.org/2003/216

[4] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: Improved definitions and efficient constructions,” in *Proc. ACM Conf. Comput. Commun. Secur.*, 2006, pp. 79–88.

[5] J. Li, Y. Shi, and Y. Zhang, “Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage,” *Int. J. Commun. Syst.*, vol. 30, no. 1, pp. 1–13, 2017.

[6] Y. Miao, J. Ma, X. Liu, X. Li, Q. Jiang, and J. Zhang, “Attribute-based keyword search over hierarchical data in cloud ,” *IEEE Trans. Serv. Comput.*, to be published.

[7] Sunil S Khatal, Data Security using KAC for Sharing Scalable Data