DABKS: Dynamic Attribute-based Keyword Search in Cloud Computing

Baishuang Hu[†], Qin Liu[†‡*], Xuhui Liu[†], Tao Peng[§], Guojun Wang[¶], and Jie Wu[‖]
[†]College of Computer Science and Electronic Engineering, Hunan University, P. R. China, 410082
[‡]State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China, 100876
[§] School of Information Science and Engineering, Central South University, P. R. China, 410083
[¶] School of Computer Science and Educational Software, Guangzhou University, P. R. China, 510006
[‖] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
[*]Correspondence to: gracelq628@hnu.edu.cn

***Abstract*—Due to its fast deployment and scalability, cloud computing has become a significant technology trend. Organizations with limited budgets can achieve great flexibility at a low price by outsourcing their data and query services to the cloud. Since the cloud is outside the organization's trusted domain, existing research suggests encrypting data before outsourcing to preserve user privacy. Two main problems that the cloud user faces while searching over encrypted data are *how to achieve a fine-grained search authorization* and *how to efficiently update the search permission*. The existing attribute-based keyword search (ABKS) scheme addresses the first problem, which allows a data owner to control the search of the outsourced encrypted data according to an access policy. This paper proposes a dynamic attribute-based keyword search (DABKS) scheme that incorporates proxy re-encryption (PRE) and a secret sharing scheme (SSS) into ABKS. The DABKS scheme, which allows the data owner to delegate policy updating operations to the cloud, takes full advantage of cloud resources. We conduct experiments on real data sets to validate the effectiveness and efficiency of our proposed scheme.**

***Index Terms*—cloud computing, searchable encryption, fine-grained search authorization, dynamic access policy.**

I. Introduction

Cloud computing, which offers computer resources dynamically via Internet, provides users with many benefits, such as fast deployment, scalability and elasticity [1], [2]. Small and medium-sized enterprises in particular can achieve great flexibility at a low price by outsourcing their data and query services to the cloud. Since the cloud service provider (CSP) may leak users' private data consciously or unconsciously, existing research [3], [4] suggests encrypting data before outsourcing to preserve user privacy.

However, data encryption would make searching over ciphertexts a very challenging task. A simple solution which downloads the whole data set from the cloud would incur extensive communication costs. Therefore, searchable encryption (SE) [5]–[9] was proposed to enable a user to retrieve data of interest while maintaining user privacy. Depending on the selection of *index key* (pk) and *search key* (sk), existing SE solutions can be classified as symmetric-key settings ($sk = pk$) or public-key settings ($sk \neq pk$). In a typical cloud computing environment, the data owner encrypts each file with traditional symmetric/asymmetric cryptography and encrypts the relevant keywords with SE under pk separately. To retrieve files described by keyword w, the data user first asks the data owner for sk, with which a *search token* for w is generated. On receiving the search request, the CSP will evaluate the search token on keyword ciphertexts and return all the matched files to the user, who will then perform decryption locally to recover file contents.

Previous SE solutions grant search permission in a *coarse-grained* way; by using the search key, the data user has the ability to generate search tokens for all the keywords. In many situations, such a search authorization would potentially risk privacy disclosure. To illustrate, let us consider the following scenario: Company A outsources its file management system to Amazon S3 for easy access by its staff. Suppose that the collaboration agreement, F, described with keywords "Company B" and "Project X" can be accessed only by the manager of Company A. If attacker Alice is allowed to first search with keyword "Project X" and then with keyword "Company B", the search results returned allows her to infer that Company A is cooperating with Company B on Project X, even if she cannot recover file content.

To alleviate this problem, Bao et al. [10] proposed an authorized searchable encryption in a multi-users setting, which allowed the data owner to enforce an access policy by distributing some secret keys to authorized users. Li et al. [11] constructed an authorized private keyword search scheme based on hierarchical predicate encryption [12]. Among others, Zheng et al. [13] proposed the attribute-based keyword search (ABKS) scheme, which utilized attribute-based encryption (ABE) [14] to achieve *fine-grained search authorization* for public-key settings. In ABKS, each keyword w_i is associated with an access policy AP, and each search token is associated with a keyword w_j and a set of attributes S. The data user can search the file only when her attributes satisfy the access policy, denoted as $S \preceq AP$, and $w_j = w_i$. However, ABKS never considered the problem of a dynamic access policy for keywords. If AP is changed to AP', the data owner needs to re-encrypt the relevant keywords with AP' so that only the users whose attributes satisfy AP' have search permission. For frequent updates on a large number of files, the workload on

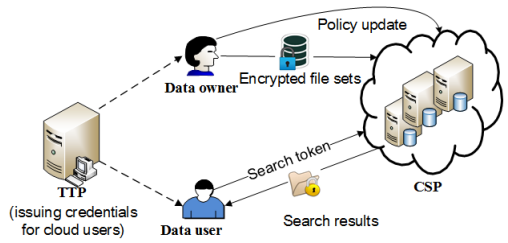


Fig. 1. System model.

the data owner is heavy.

In this paper, we propose a dynamic attribute-based keyword search (DABKS) scheme by incorporating proxy re-encryption (PRE) [15] and a secret sharing scheme (SSS) into ABKS. In DABKS, the CSP can update the access policy for keywords without compromising user privacy. Specifically, DABKS expresses the access policy AP as an access tree and transforms the problem of updating an AND/OR gate in AP to that of updating a threshold gate. For example, the AND gate is transformed to (t, t) gate, and the OR gate is transformed to $(1, t)$ gate. Therefore, the updating of the AND gate can be treated as updating (t, t) gate to (t', t') gate, and the updating of the OR gate can be treated as updating $(1, t)$ gate to $(1, t')$ gate, where $t' = t + 1$ for adding an attribute to the AND/OR gate and $t' = t - 1$ for removing an attributes from the AND/OR gate. Our key contributions are as follows:

1) To the best of our knowledge, this is the first attempt made to devise a dynamic access policy for fine-grained search authorization in a cloud environment.
2) The proposed DABKS scheme can largely reduce the workload on the data owner by delegating the policy updating operations to the cloud.
3) We conduct experiments on real data sets to validate the effectiveness and efficiency of our proposed scheme.

II. Preliminary

A. System Model

As shown in Fig. 1, the system is composed of the following parties: the cloud users, the cloud service provider (CSP), and a trusted third party (TTP). The cloud users, who pay the services residing on the cloud or deploy their own applications/systems in the cloud, can be further classified into *data owner* and *data user*. The data owner outsources the encrypted data and keywords to the cloud and authorizes multiple data users to access them. The data user will retrieve data of interest according to a keyword-based search. The CSP operates the cloud platforms, which provide not only the storage and search services, but also perform policy updating operations on behalf of the data owner. The TTP is responsible for issuing credentials to all cloud users.

B. Attack Model

The cloud users and the TTP are assumed to be fully trusted. The CSP is assumed to be *honest but curious*; this means it will always correctly execute a given protocol, but may try to learn some additional information about the stored data and the received message. Furthermore, the communication channels are assumed to be secured under existing security protocols such as SSL and SSH.

While file content privacy can be achieved by approaches proposed in [3], [16], the DABKS scheme aims to preserve *keyword privacy* and *query privacy* for the cloud users. However, in the public-key setting, the attacker can handpick a keyword to encrypt and can check whether the resulting ciphertext and the target search token correspond to the same keyword. Thus, it is impossible to protect search tokens from the keyword-guessing attack. Let a probabilistic polynomial-time adversary $\mathcal{A}$ model the CSP. Following the work in [13], our scheme is considered secure if the following holds:

1) **Selective security against chosen-keyword attacks.** Given a keyword ciphertext that mismatches all search tokens, adversary $\mathcal{A}$ cannot infer any information about the keyword plaintext in the selective security model [17] and cannot deduce any information about the file content. 2) **Keyword secrecy.** The probability that adversary $\mathcal{A}$ learns the keyword from the keyword ciphertext and search tokens is negligibly more than the probability of a correct random-keyword guess.

C. Access Tree

As the work in [14] suggests, the access policy AP can be depicted as an access tree $\mathcal{T}$ where each interior node denotes a gate and each leave node is depicted as an attribute. In $\mathcal{T}$, each node x is associated with a threshold value k_x. For the interior node x with N_x children, $k_x = 1$ when x is an OR gate, and $k_x = N_x$ when x is an AND gate. For all leave nodes, the threshold value is 1. Let $lev(\mathcal{T})$ denote leave nodes in $\mathcal{T}$. If $x \in lev(\mathcal{T})$, $att(x)$ is used to denote the attribute associated with node x. Furthermore, $\mathcal{T}$ defines an ordering between the children of each node, and $parent(x)$ and $index(x)$ return the parent and the order number of children node x, respectively.

Let $\mathcal{T}$ with root R correspond to the access tree of access policy AP. To check whether a set of attributes S_u satisfies AP, denoted as $S_u \preceq AP$, we compute $\mathcal{T}_R(S_u)$ recursively as follows: Suppose that $\mathcal{T}_x$ denotes the subtree of $\mathcal{T}$ rooted at the node x. If x is a non-leaf node, we evaluate $\mathcal{T}_b(S_u)$ for each child b of node x. $\mathcal{T}_x(S_u)$ returns 1 if and only if at least k_x children return 1. If x is a leaf node, then $\mathcal{T}_x(S_u)$ returns 1 if and only if $att(x) \in S_u$. To share the secret σ in $\mathcal{T}$, the SSS generates $\Delta = \{q_x(0)\}_{x \in \mathcal{T}}$ as follows:

$SSS(\sigma, \mathcal{T}) \rightarrow \Delta$. A random polynomial q_R of degree $k_R - 1$ is chosen for $q_R(0) = \sigma$. The rest of the points in q_R are randomly chosen. For each node $x \in \mathcal{T}$, a random polynomial q_x of degree $k_x - 1$ is chosen for $q_x(0) = q_{parent(x)}(index(x))$. The rest of points in q_x are chosen randomly. To recover the secret σ, the users with sufficient secret shares can perform Lagrange interpolation recursively. Please refer to [14] for more details.

III. The DABKS scheme

Let $\mathbb{A} = \{A_1, \ldots, A_M\}$ denote the universal attributes in the system. The data user u is described by a set of attributes $S_u \subseteq \mathbb{A}$. The data owner v holds a collection of files $\Omega = \{F_1, \ldots, F_n\}$, where each file F_i can be described by a set of

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
PK, MK	System public/master key
S_u	Attribute set associated with data user u
sk_u	Search key for data user u
$\mathcal{T}_w, \mathcal{T}'_w$	Original/new access policy for keyword w
cph_w, cph'_w	Original/new ciphertext for keyword w
Tok_w	Search token for keyword w
Δ	A set of secret shares for nodes in $\mathcal{T}_w$
UK	Update key for updating $\mathcal{T}_w$ to $\mathcal{T}'_w$
Φ	Auxiliary information for Att2AND/Att2OR gate

distinct keywords W_i. Before uploading file F_i to the cloud, v will first encrypt F_i with ABE under access policy AP_F and will then encrypt each keyword in W_i with DABKS under access policy AP_K. The ciphertext for keyword w and for file F_i is denoted as cph_w and C_{F_i}, respectively. It is worth noting that AP_F stipulating which entities can decrypt F_i and AP_K stipulating which entities can search over W_i may be different. For ease of illustration, we assume that the access policies for W_i and F_i are the same, i.e., $AP_F = AP_K$. To retrieve files containing keyword w, the data user will issue a search token Tok_w to the CSP, which will return $\{\{cph_w\}_{w\in W_i}, C_{F_i}\}$ only when $S_u \preceq AP_K$ and $w \in W_i$. To update AP_K to AP'_K, the data owner will send an update instruction Γ to the CSP, which will update related keyword ciphertexts to preserve the correctness of the access policy for keywords W_i. For quick reference, the most used notations are shown in Table I.

A. Construction

Fig. 2 shows the working process of the DABKS scheme, where algorithms *GenKey*, *EncFile*, and *Decrypt* related to ABE [14] are used to preserve file privacy. Since our work focuses on preserving keyword privacy and query privacy, we omit the construction of these algorithms in this paper.

(Initialization phase)

•$Init(\lambda) \rightarrow (PK, MK)$: Let $e : \mathbf{G}_0 \times \mathbf{G}_0 \rightarrow \mathbf{G}_1$ be the bilinear group, where $\mathbf{G}_0$ and $\mathbf{G}_1$ are cyclic groups of prime order p, the TTP takes security parameter λ as input, and it sets the system public key PK and the system master key MK as follows:

$$PK = \{H_1, H_2, e, g, p, g^a, g^b, g^c\}, \quad MK = (a, b, c),$$

where $H_1 : \{0,1\}^* \rightarrow \mathbf{G}_0$ is a hash function modeled as a random oracle, $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p$ is a one-way hash function, $g \in \mathbf{G}_0$ is the random generator of $\mathbf{G}_0$, and a, b, c are randomly chosen from $\mathbb{Z}_p$.

•$KeyGen(PK, MK, S_u) \rightarrow sk_u$: For data user u associated with attribute set S_u, the TTP generates search key sk_u for u as follows: it first randomly selects $r \in \mathbb{Z}_p$, and computes $D = g^{(ac-r)/b}$. Then, it selects a random $r_j \in \mathbb{Z}_p$ for each attribute $A_j \in S_u$ and computes $B_j = g^r H_1(A_j)^{r_j}$ and $\bar{B}_j = g^{r_j}$. The search key is set as follows:

$$sk_u = (S_u, D, \{(B_j, \bar{B}_j)\}_{A_j \in S_u})$$

(Store phase)

•$EncKW(PK, w, \mathcal{T}_w) \rightarrow (cph_w, \Delta)$: To encrypt keyword w under access tree $\mathcal{T}_w$, the data owner first randomly selects $r_1, \sigma \in \mathbb{Z}_p$ and computes $K_1 = g^{cr_1}$, $K_2 = g^{a(r_1+\sigma)} g^{bH_2(w)r_1}$, and $K_3 = g^{b\sigma}$. Then, it computes $C_{x_i} = g^{q_{x_i}(0)}$ and $\bar{C}_{x_i} = H_1(att(x_i))^{q_{x_i}(0)}$, where $q_{x_i}(0)$ is the share of secret σ for leave node x_i in $\mathcal{T}_w$ generated by SSS. The ciphertext for keyword w is set as the following:

$$cph_w = (\mathcal{T}_w, K_1, K_2, K_3, \{C_i = (C_{x_i}, \bar{C}_{x_i})\}_{x_i \in lev(\mathcal{T}_w)})$$

For updating access tree $\mathcal{T}_w$ to $\mathcal{T}'_w$, the data owner needs to preserve $\Delta = \{q_{x_i}(0)\}_{x_i \in \mathcal{T}_w}$ generated by SSS.

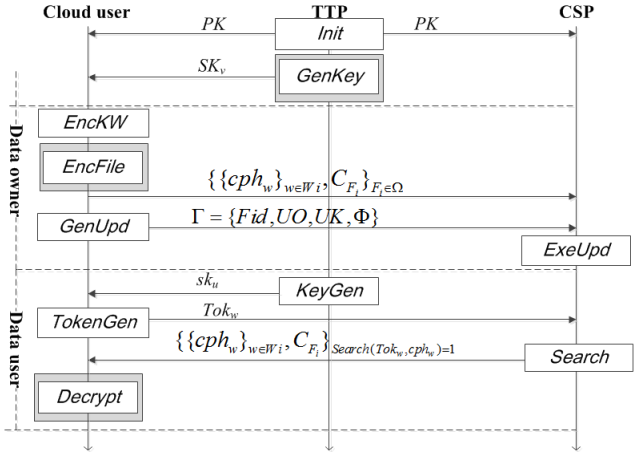


Fig. 2. Working process of the DABKS scheme.

(Search phase)

•$TokenGen(sk_u, w) \rightarrow Tok_w$: To retrieve files containing keyword w, data user u associated with attribute set S_u chooses a random $s \in \mathbb{Z}_p$, and computes $tk_1 = (g^a g^{bH_2(w)})^s$, $tk_2 = g^{cs}$, and $tk_3 = D^s = g^{(ac-r)s/b}$. In addition, for each $A_j \in S_u$, it computes $B'_j = B_j^s$ and $\bar{B}'_j = \bar{B}_j^s$. The search token for keyword w is set as:

$$Tok_w = (S_u, tk_1, tk_2, tk_3, \{(B'_j, \bar{B}'_j)\}_{A_j \in S_u}).$$

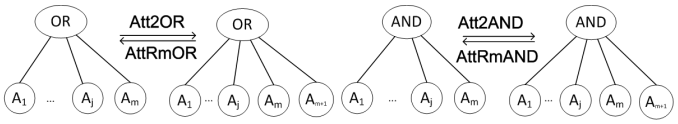
•$Search(Tok_w, cph_w) \rightarrow \{0,1\}$: On receiving the search token Tok_w from data user u, the CSP first constructs a set $S \in S_u$ that satisfies the access tree $\mathcal{T}_w$ specified in cph_w, and then it computes $E_{x_i} = e(B'_i, C_{x_i})/e(\bar{B}'_i, \bar{C}_{x_i}) = e(g,g)^{rsq_{x_i}(0)}$ for each attribute $A_i \in S$, where $A_i = att(x_i)$ for $x_i \in lev(\mathcal{T})$. Next, it executes the Lagrange interpolation to recover $E_R = e(g,g)^{rs\sigma}$. Finally, it tests whether Eq. 1 holds. If so, it outputs 1. Otherwise, 0 is the output.

$$e(K_2, tk_2) = e(K_1, tk_1) e(tk_3, K_3) E_R \tag{1}$$

(Update phase)

•$GenUpd(\Delta, \mathcal{T}'_w) \rightarrow (UK, \Phi)$: Given the new access tree $\mathcal{T}'_w$ for keyword w, the data owner first locates the gate node that will be modified. Let node y denote the AND/OR gate being updated, where $A_1, \ldots, A_m$ and $A_1, \ldots, A_{m'}$ are original and new attributes under node y, respectively. Given $q_y(0)$, the share associated with node y, it first takes $q_x(0)$ and $\mathcal{T}'_w$ as inputs of the SSS algorithm and next obtains the new secret shares for nodes $x_1, \ldots, x_{m'}$ in $\mathcal{T}'_w$, denoted as $\{q'_{x_i}(0)\}_{i\in[1,m']}$. Then, it generates an update key for attributes $A_1, \ldots, A_m$ as follows:

$$UK = \{(UK_{1,i}, UK_{2,i})\}_{i\in[1,m]} \tag{2}$$



(a)Att2OR and AttRmOR (b)Att2AND and AttRmAND

Fig. 3. Policy updating operations.

where $UK_{1,i} = g^{\sigma}$, $UK_{2,i} = H_1(att(x_i))^{\sigma}$, and $\sigma = q'_{x_i}(0) - q_{x_i}(0)$. Furthermore, for adding an attribute A_{m+1} under gate node y, the data owner generates the new ciphertext C'_{m+1} for A_{m+1} as follows: it computes $C'_{x_{m+1}} = g^{q'_{x_{m+1}}(0)}$ and $\bar{C}'_{x_{m+1}} = H_1(att(x_{m+1}))^{q'_{x_{m+1}}(0))}$. The new ciphertext is set as $C'_{m+1} = (C'_{x_{m+1}}, \bar{C}'_{x_{m+1}})$ which will be used as the auxiliary information Φ.

•$ExeUpd(UK, \Phi, cph_w) \rightarrow cph'_w$: After receiving the policy updating request from the data owner, the CSP utilizes the update key UK to update the original keyword ciphertext cph_w as follows: for each leave node x_i under the changing gate node y, it computes $C'_{x_i} = C_{x_i} \cdot UK_{1,i} = g^{q'_{x_i}(0)}$ and $\bar{C}'_{x_i} = \bar{C}_{x_i} \cdot UK_{2,i} = H_1(att(x_i))^{q'_{x_i}(0)}$. Then, the new ciphertext for keyword w is set as:

$$cph'_w = (\mathcal{T}'_w, K_1, K_2, K_3, \{C'_i = (C'_{x_i}, \bar{C}'_{x_i})\}_{x_i \in lev(\mathcal{T}'_w)})$$

B. Policy Updating

The DABKS scheme achieves an efficient update of access policy by delegating policy updating operations to the cloud as follows: if file F_i's access policy is changed to AP', the data owner first builds a new access tree $\mathcal{T}'_w$ for AP' and then generates the update key UK and some auxiliary information Φ. The update instruction sent to the CSP is set to $\Gamma = \{Fid, UO, UK, \Phi\}$, where Fid is the ID of file F_i and UO is the specific update operation. On receiving the policy update request, the CSP locates F_i and generates new ciphertexts $\{cph'_w\}_{w \in W_i}$ for W_i based on Γ.

Inspired by the work in [18], we consider four basic operations involved in policy updating (Fig. 3): $Att2OR$ denotes adding an attribute to an OR gate, $Att2AND$ denotes adding an attribute to an AND gate, $AttRmOR$ denotes removing an attribute from an OR gate, and $AttRmAND$ denotes removing an attribute from an AND gate. Let node y be the AND/OR gate that will be updated where $A_1, \ldots, A_m$ are the original attributes under y. Let $q_y(0)$ and $\{q_{x_1}(0), \ldots, q_{x_m}(0)\}$ denote the secret shares for node y and y's children nodes $x_1, \ldots, x_m$, where $att(x_i) = A_i$ for $i \in [1, m]$. Given an access tree $\mathcal{T}_w$, our scheme will produce a ciphertext for each leave node x based on share $q_x(0)$. The original and new ciphertexts for node x_i are denoted as C_i and C'_i, respectively.

•$Att2OR$: This operation can be transformed to updating a $(1, m)$ gate to a $(1, m+1)$ gate. Given $q_y(0)$ and the new access policy $\mathcal{T}'$, the data owner runs SSS to generate new shares $\{q'_{x_1}(0), \ldots, q'_{x_{m+1}}(0)\}$ for attributes $A_1, \ldots, A_{m+1}$. Since $q_{x_i}(0) = q'_{x_i}(0) = q_y(0)$ for $i \in [1, m+1]$, the ciphertexts for the original attributes will not be changed, i.e., $C'_i = C_i$ for $i \in [1, m]$. For the newly added attribute A_{m+1}, the data owner needs to generate a new ciphertext C'_{m+1} based on $q'_{x_{m+1}}(0)$. Finally, it sets the update instruction $\Gamma = \{Fid, Att2OR, NULL, C'_{m+1}\}$ to the CSP, which will add C'_{m+1} to cph_w and update the access tree to $\mathcal{T}'_w$ by adding A_{m+1} under node y.

•$AttRmOR$: This operation can be transformed to updating a $(1, m)$ gate to a $(1, m-1)$ gate. As in the $Att2OR$ operation, we have $q_{x_i}(0) = q'_{x_i}(0) = q_y(0)$ for $i \in [1, m-1]$. Therefore, the data owner will send $\Gamma = \{Fid, AttRmOR, NULL, NULL\}$ to the CSP, which will remove C_m from cph_w and update the access tree to $\mathcal{T}'_w$ by removing A_m under node y.

•$Att2AND$: This operation can be transformed to updating a (m, m) gate to a $(m+1, m+1)$ gate. Given $q_y(0)$ and the new access policy $\mathcal{T}'$, the data owner runs SSS to generate new shares $\{q'_{x_1}(0), \ldots, q'_{x_{m+1}}(0)\}$ for attributes $A_1, \ldots, A_{m+1}$. Next, it executes the $GenUpd$ algorithm to generate the update key UK for attributes $A_1, \ldots, A_m$. Moreover, it generates a ciphertext C'_{m+1} for the newly added attribute A_{m+1} based on $q'_{x_{m+1}}(0)$. Finally, it sends $\Gamma = \{Fid, Att2AND, UK, C'_{m+1}\}$ to the CSP, which will execute the $ExeUpd$ algorithm to update the ciphertext C_i to C'_i for $i \in [1, m]$, add the new ciphertext C'_{m+1} to the cph_w, and update the access tree to $\mathcal{T}'_w$ by adding A_{m+1} under node y.

•$AttRmAND$: This operation can be transformed to updating a (m, m) gate to a $(m-1, m-1)$ gate. Given $q_y(0)$ and the new access policy $\mathcal{T}'$, the data owner runs SSS to generate new shares $\{q'_{x_1}(0), \ldots, q'_{x_{m-1}}(0)\}$ for attributes $A_1, \ldots, A_{m-1}$. Next, it executes the $GenUpd$ algorithm to generate the update key UK for $C_1 \ldots, C_{m-1}$. Finally, it sends $\Gamma = \{Fid, AttRmAND, UK, NULL\}$ to the CSP, which executes the $ExeUpd$ algorithm update of the ciphertext C_i to C'_i for $i \in [1, m-1]$, removes C_m from cph_w, and updates the access tree to $\mathcal{T}'_w$ by removing A_m under node y.

C. Correctness and Security Sketch

Correctness sketch. In the *Search* algorithm, for each attribute $A_j \in S$, we have the following:

$$\begin{aligned} E_{x_i} &= \frac{e(B'_j, C_{x_i})}{e(\bar{B}'_j, \bar{C}_{x_i})} = \frac{e(g^{rs} H_1(A_j)^{r_j s}, g^{q_{x_i}(0)})}{e(g^{r_j s}, H_1(att(x_i))^{q_{x_i}(0)})} \\ &= e(g,g)^{rs q_{x_i}(0)} \end{aligned} \tag{3}$$

If $S_u \preceq \mathcal{T}_w$, we can recover $E_R = e(g,g)^{rs q_R(0)} = e(g,g)^{rs\sigma}$ by executing the Lagrange interpolation. Therefore, the right side of Eq. 1 will evolve as follows:

$$\begin{aligned} & e(K_1, tk_1) e(tk_3, K_3) E_R \\ &= e(g^{cr_1}, g^{as} g^{bsH_2(w)}) e(g^{(acs-rs)/b}, g^{b\sigma}) e(g,g)^{rs\sigma} \\ &= e(g,g)^{acs(r_1+\sigma)} e(g,g)^{bcsH_2(w)r_1} \end{aligned} \tag{4}$$

The left side of Eq. 1 will evolve as follows:

$$\begin{aligned} e(K_2, tk_2) &= e(g^{a(r_1+\sigma)} g^{bH_2(w)r_1}, g^{cs}) \\ &= e(g,g)^{acs(r_1+\sigma)} e(g,g)^{bcsH_2(w)r_1} \end{aligned} \tag{5}$$

Therefore, Eq. 1 holds only when $S_u \preceq \mathcal{T}_w$.

Then, we prove that the output of the *Search* algorithm is still correct after the *GenUpd* and *ExeUpd* algorithms. Due to limited space, we only provide the correct proof for the *Att2AND* operation in this paper. Let $q'_{x_1}, \ldots, q'_{x_m}, q'_{x_{m+1}}$ be the secret shares of σ for the leave nodes in $\mathcal{T}'_w$. The original

TABLE II
PERFORMANCE ANALYSIS OF DABKS

	Computation	Communication
KeyGen	$(2S+2)E_0 + SH_1$	$(2S+1)\|\mathbf{G}_0\|$
EncKW	$(2N+4)E_0 + NH_1$	$(2N+3)\|\mathbf{G}_0\|$
TokenGen	$(2S+4)E_0$	$(2S+3)\|\mathbf{G}_0\|$
Search	$(2N+3)e + NE_1$	
GenUpd(Att2AND)	$(2E_0+H_1)(m+1)$	$2(m+1)\|\mathbf{G}_0\|$
GenUpd(AttRmAND)	$(2E_0+H_1)(m-1)$	$2(m-1)\|\mathbf{G}_0\|$

attribute A_i where $i \in [1,m]$, $C'_i = (C'_{x_i}, \bar{C}'_{x_i})$ will be constructed as follows:

$$\begin{aligned} C'_{x_i} &= C_{x_i} \cdot UK_{1,i} \\ &= g^{q_{x_i}(0)} g^{q'_{x_i}(0) - q_{x_i}(0)} = g^{q'_{x_i}(0)} \\ \bar{C}'_{x_i} &= \bar{C}_{x_i} \cdot UK_{2,i} \\ &= H_1(att(x_i))^{q_{x_i}(0)} H_1(att(x_i))^{q'_{x_i}(0) - q_{x_i}(0)} \\ &= H_1(att(x_i))^{q'_{x_i}(0)} \end{aligned} \tag{6}$$

The new ciphertext C'_{m+1} corresponding to A_{m+1} is constructed as follows:

$$\begin{aligned} C'_{m+1} &= (C'_{x_{m+1}}, \bar{C}'_{x_{m+1}}) \\ &= (g^{q'_{x_{m+1}}(0)}, H_1(att(x_{m+1}))^{q'_{x_{m+1}}(0)}) \end{aligned} \tag{7}$$

Given the updated ciphertexts C'_i for $i \in [1, m+1]$, $E_R = e(g,g)^{rsq_R(0)} = e(g,g)^{rs\sigma}$ can be recovered only when $S_u \preceq \mathcal{T}_w$. Therefore, the DABKS scheme is correct. ■

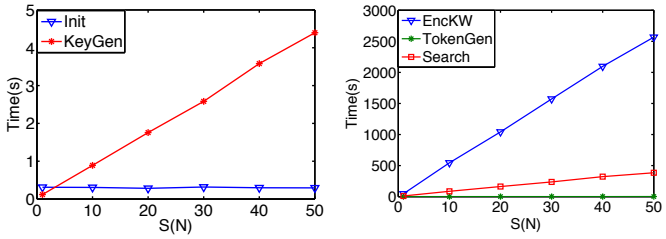
Security sketch. The work in [13] has proven that given the one-way hash function H_2, the ABKS scheme is selectively secure against chosen-keyword attacks in the generic bilinear group model and achieves keyword secrecy in the random oracle model. The correctness proof has proven that the DABKS scheme carries out a correct search control after the update of access policy AP. Therefore, the security of our scheme can be derived from that of the ABKS scheme. ■

IV. EVALUATION

A. Performance Analysis

We will analyze the performance of the DABKS scheme in terms of computational and communication complexity. For ease of understanding, we provide the following notations to denote the running time for various operations in our scheme: H_1 is used to denote the operation of mapping a bit-string to an element of $\mathbf{G}_0$, e is used to denote the pairing operation, E_0 and E_1 are used to denote the exponentiation operation in $\mathbf{G}_0$ and $\mathbf{G}_1$, respectively. We neglect the multiplication in $\mathbf{G}_0$, $\mathbf{G}_1$ and hash operations since they are much less expensive than the above operations.

Table II shows the complexity of the DABKS scheme, where S denotes the number of attributes associated with a data user, N denotes the number of attributes in a data owner's access policy, m denotes the number of attributes under an AND/OR gate node which will be updated, and $|\mathbf{G}_0|$ denotes the length of elements in $\mathbf{G}_0$. The *Init* algorithm will be run in the system initialization phase. Therefore, the TTP spends most of its time generating a search key for the data users. The complexity of generating such a search key relates to S. For the data owner, the cost of the *EncKW* algorithm will grow linearly with N, and the cost of the *GenUpd* algorithm related to the AND gate will grow linearly with m. For the OR gate, the cost of the *GenUpd* algorithm is almost 0. For the data user, the cost of the *TokenGen* algorithm will grow linearly with S. For the CSP, the *ExeUpd* algorithm involves only the multiplication operations, the cost of which can be neglected.



(a) Initialization phase (b) Store and search phases

Fig. 4. Execution time (in seconds) of our scheme.

B. Parameter Setting

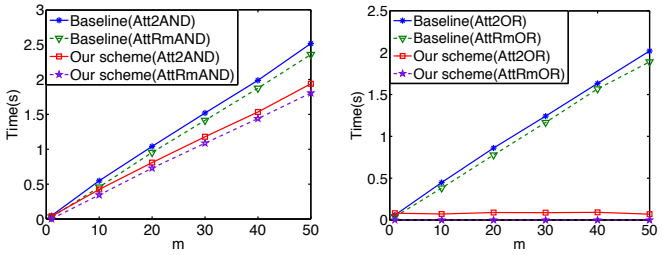
Experiments are conducted on a local machine running the Microsoft Windows 7 Ultimate operating system with an Inter Core i5 CPU running at 3.2GHz and 8GB memory. The programs are implemented in Java and compiled using Eclipse 4.6.0. The cryptographic algorithms in the verification phase are implemented with JPBC library [19]. We use a 160-bit elliptic curve $y^2 = x^3 + x$ over a 512-bit finite field, in which p is a 160-bit length prime, and the length of element in $\mathbf{G}_0$ is 512-bit. To validate the effectiveness and efficiency of the DABKS scheme, we conduct a performance evaluation on a real data set, the Internet Request For Comments(RFC) [20]. This dataset has 6,870 plaintext files with a total size about 349MB.We use Hermetic Word Frequency Counter [21] to extract keywords from each RFC file, and choose $[5, 10]$ keywords for each file after ranking them by frequency of occurrence. In the experiments, we select 1,000 files from the data set, where the number of distinct keywords is 1,307.

The parameter settings in the experiments are as follows: the number of attributes that are involved in an access policy N ranges from 1 to 50, the number of data users' attributes S ranges from 1 to 50, and the number of attributes under a updating gate node m ranges from 1 to 50. In our implementation, we set the example access policy as $AP = (A_1 \wedge A_2 \wedge \ldots \wedge A_N)$, encrypt each keyword with the same access policy, and set $S = N$.

C. Experiment Results

Fig. 4-(a) shows the execution time of the initialization phase. We observe that when S ranges from 1 to 50 the *Init* algorithm costs about 0.32s and that the *KeyGen* algorithm costs from 0.12s to 4.43s. The *Init* algorithm will be executed when the system is initialized, and the *KeyGen* algorithm will be run while a new cloud user joins the system. Therefore, the workload on the TTP is relatively low.

In the store phase, the *EncKW* algorithm needs to encrypt 1,307 distinct keywords extracted from 1,000 data files, and in the search phase, the *Search* algorithm needs to to perform search operations over a collection of keyword ciphertexts to find the matching results. In Fig. 4-(b), we observe that the



(a) $Att2AND$ and $AttRmAND$ (b) $Att2OR$ and $AttRmOR$

Fig. 5. Execution time (in seconds) for *GenUpd*.

EncKW algorithm run by the data owner incurs the maximal execution time and the *TokenGen* algorithm run by the data user incurs the minimum execution time. For example, while N ranges from 1 to 50, the execution time grows from 46.25s to 2,567.23s for *EncKW*, from 0.05s to 0.93s for *TokenGen*, and from 10.92s to 384.47s for *Search*. However, the *EncKW* algorithm is executed just once to encrypt all the keywords, while the *Search* and *TokenGen* algorithms are executed as many times as needed to conduct searches.

The execution time of the *GenUpd* algorithm is shown in Fig. 5. In ABKS [13], once the access policy is changed, the data owner needs to re-encrypt the relevant keywords with the new access policy and to send the new ciphertexts to the cloud. The *baseline* denotes the re-encryption cost on the data owner in [13]. For the *Att2AND* and *AttRmAND* operations, the access policy is set as $AP = (A_1 \wedge A_2 \wedge \ldots \wedge A_m)$. The execution time of *Att2AND* in our scheme increases from 0.028s to 2.36s, and the execution time of *Att2AND* in *baseline* increases from 0.048s to 2.52s as m increases from 1 to 50. The comparison results of *AttRmAND* are consistent with those of *Att2AND*. For the *Att2OR* and *AttRmOR* operations, the access policy is set as $AP = (A_1 \vee A_2 \vee \ldots \vee A_m)$. Since the secret share of the remaining attributes is not changed in our scheme, the incurred time on the data owner gets close to 0. However, as m increases from 1 to 50, the execution time of *baseline* increases from 0.058s to 2.02s for *Att2OR* and from 0.038s to 1.89s for *AttRmOR*. Therefore, our scheme incurs fewer computation costs on the data owner compared with the *baseline*. During the update phase, the CSP needs to execute the *ExeUpd* algorithm to update the keyword ciphertexts. For updating the AND gate, the execution time of the *ExeUpd* algorithm increases from 0.18ms to 8.6ms while m increases from 1 to 50; for updating the OR gate, the *ExeUpd* algorithm has almost zero execution time. The experiment results prove that the cloud user should outsource the keyword search and policy update operations to the CSP in order to take full advantage of the CSP's vast computation capabilities. Therefore, the workload of the data user will be largely reduced.

V. Conclusion

In this paper, we propose a DABKS scheme to simultaneously achieve fine-grained search authorization and efficient update of access policy. Our scheme takes full advantage of cloud resources by delegating policy update operations to the CSP. Experiment results verify its feasibility and effectiveness. However, the DABKS scheme supports only the single-keyword search. As part of our future work, we will try to extend our scheme to a multi-keyword search scenario, which supports conjunctive, subset, and range queries on encrypted outsourced data.

Acknowledgment

This work was supported in part by NSFC grants 61632009 and 61402161; NSF grants CNS 1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, and ECCS 1231461; the Hunan Provincial Natural Science Foundation of China grant 2015JJ3046; and the Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) under Grant No. SKLNST-2016-2-20.

References

[1] Q. Liu, C. C. Tan, J. Wu, and G. Wang, "Towards differential query services in cost-efficient clouds," *IEEE Transactions on Parallel and Distributed Systems*, 2014.
[2] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, 2016.
[3] Q. Liu, G. Wang, and J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," *Information Sciences*, 2014.
[4] J. Li, J. Li, X. Chen, C. Jia, and W. Lou, "Identity-based encryption with outsourced revocation in cloud computing," *IEEE Transactions on Computers*, 2015.
[5] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE S&P*, 2000.
[6] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of Eurocrypt*, 2004.
[7] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on parallel and distributed systems*, 2014.
[8] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. of IEEE INFOCOM*, 2014.
[9] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," 2015.
[10] F. Bao, R. H. Deng, X. Ding, and Y. Yang, "Private query on encrypted data in multi-user settings," in *Information Security Practice and Experience*. Springer, 2008.
[11] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted personal health records in cloud computing," in *Proc. of IEEE ICDCS*, 2011.
[12] T. Okamoto and K. Takashima, "Hierarchical predicate encryption for inner-products," in *Proc. of ASIACRYPT*, 2009.
[13] Q. Zheng, S. Xu, and G. Ateniese, "VABKS: verifiable attribute-based keyword search over outsourced encrypted data," in *Proc of IEEE INFOCOM*, 2014.
[14] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. of IEEE S&P*, 2007.
[15] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. of EUROCRYPT*, 1998.
[16] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *Proc. of ACM CCS*, 2010.
[17] R. Canetti, S. Halevi, and J. Katz, "Chosen-ciphertext security from identity-based encryption," in *Proc. of EUROCRYPT*, 2004.
[18] K. Yang, X. Jia, K. Ren, R. Xie, and L. Huang, "Enabling efficient access control with dynamic policy updating for big data in the cloud," in *Proc. of IEEE INFOCOM*, 2014.
[19] A. De Caro and V. Iovino, "jpbc: Java pairing based cryptography," in *Proc. of IEEE ISCC*, 2011.
[20] RFC, "Request for comments database," *http://www.ietf.org/rfc.html*.
[21] HERMETIC, "Hermetic word frequency counter," *http://www.hermetic.ch/wfc/wfc.htm*.