Keyword Search With Access Control Over Encrypted Cloud Data

Zhirong Shen, *Member, IEEE*, Jiwu Shu, *Member, IEEE*, and Wei Xue, *Member, IEEE*

***Abstract*— In this paper, we study the problem of keyword search with access control (KSAC) over encrypted data in cloud computing. We first propose a scalable framework where user can use his attribute values and a search query to locally derive a search capability, and a file can be retrieved only when its keywords match the query and the user's attribute values can pass the policy check. Using this framework, we propose a novel scheme called KSAC, which enables keyword search with access control over encrypted data. KSAC utilizes a recent cryptographic primitive called hierarchical predicate encryption to enforce fine-grained access control and perform multi-field query search. Meanwhile, it also supports the search capability deviation, and achieves efficient access policy update as well as keyword update without compromising data privacy. To enhance the privacy, KSAC also plants noises in the query to hide users' access privileges. Intensive evaluations on real-world dataset are conducted to validate the applicability of the proposed scheme and demonstrate its protection for user's access privilege.**

***Index Terms*— Keyword search, access control, encrypted data, hierarchical predicate encryption.**

I. Introduction

THE CLOUD has become an important platform for data storage and processing. It centralizes essentially unlimited resources (e.g., storage capacity) and delivers elastic services to end users. However, a number of challenges, including concerns about data security and users' privacy, still exist [2]–[5]. For example, a user's electronic health records are sensitive data and, if uploaded into the cloud, should not be disclosed to the cloud administrators and any other unauthorized users without data owners' permission. Thus data confidentiality protection (to hide the plaintext against unauthorized parties) and data access control (to grant user's access privilege) are usually required when storing data onto the cloud.

Encryption is a commonly used method to preserve data confidentiality. However, traditional plaintext keyword search demands to retrieve all the encrypted data files from the cloud, and perform search after data decryption. This methodology is extremely unpractical for traditional networks, especially for the wireless network (e.g., wireless sensor network and mobile network) seriously constrained by resources like energy, bandwidth, and computation capability [6], [7].

Manuscript received October 7, 2016; revised November 25, 2016; accepted November 25, 2016. Date of publication December 1, 2016; date of current version January 6, 2017. This work was supported in part by the National Key R&D Program under Grant 2016YFA0602103, in part by the Natural Science Foundation of China under Grant 61232003, Grant 61327902, Grant 91530323, Grant 61602120, and Grant 61361120098, in part by the Beijing Municipal Science and Technology Commission of China under Grant D151100000815003, in part by the Priority Academic Program Development of Jiangsu Higher Education Institutions, and in part by the Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology. This paper was presented at the 22nd IEEE/ACM Symposium on Quality of Service, 2014 [1]. The associate editor coordinating the review of this paper and approving it for publication was Prof. Subhas C. Mukhopadhyay. *(Corresponding author: Jiwu Shu.)*

The authors are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: shujw@tsinghua.edu.cn).

Digital Object Identifier 10.1109/JSEN.2016.2634018

A. Related Works on Searchable Encryption

Aiming at enabling secure and efficient search over encrypted data, *Searchable Encryption* (SE) (e.g., [6]–[15]) receives increasing attentions in recent years, in which a query is encrypted as a *search capability* and a cloud server will return files matching the query embedded in the capability, without having to know the keywords both in the capability and in file's encrypted index. The first symmetric-key-based searchable encryption scheme is proposed by Song *et al.* [10]. After that, Goh *et al.* [13] presented secure indexed over encrypted data by employing Bloom Filter. To securely process the retrieved files and make them more conform to users request, Wang *et al.* [11] introduced secure ranked keyword search based on "order-preserving encryption [16]. In the public key setting, Golle *et al.* [6] first introduced the searchable encryption scheme by using bilinear mapping [46]. Waters *et al.* [12] fulfilled searchable audit log using symmetric encryption and IBE [17] respectively. Li *et al.* [18] studied the fuzzy keyword search over encrypted cloud data by utilizing edit distance.

To support multiple keywords search, Golle *et al.* [6] considered conjunctive keyword search over encrypted data. Shi *et al.* [9] realized multi-dimensional range query over encrypted data. Shen *et al.* [19] investigated the encrypted search with preference by utilizing Lagrange polynomial and secure inner-product computation. Li *et al.* [20] considered authorized private keyword search. It only achieved LTA-level authorization which was far coarser than user level access control, and missed the protection of the users access privacy. Based on the uni-gram, Fu *et al.* [21] proposed an efficient multi-keyword fuzzy ranked search scheme with improved accuracy. To efficiently support dynamic updates, Xia *et al.* [22] constructed a special tree-based index structure by using vector space and TF×IDF model. Fu *et al.* [23] found that previous keyword-based search schemes ignored the semantic information. They then developed an semantic search scheme based on the concept hierarchy and the semantic relationship between concepts in the encrypted datasets. Zhangjie *et al.* [24] designed a searchable encryption scheme that used vector space model for multi-keyword ranked search and constructed a tree-based index to enable parallel search.

However, most of existing SE schemes assume that every user can access all the shared files. Such assumption does not hold in the cloud environment where users are actually granted different access permissions according to the access-control policy determined by data owners. Therefore, it is important to study how to efficiently enforce the access policy when searching over encrypted data.

B. Related Works on Access Control Over Encrypted Data

There have also been a number of works on access control over encrypted data. To offer fine-grained access control over encrypted shared data with lightweight key management, Bethencourt *et al.* [25] proposed CP-ABE that tied specified access policy with the encrypted data, so that only the user possessing the satisfied attributes could decrypt the data. As the dual problem of CP-ABE, KP-ABE [26] embedded the access policy in the users keys while the data were labeled with attributes. HVE [14] and PE [27] were two new tools which could be used to perform access control over encrypted data, and they all employed composite-order groups that were computationally expensive. Vimercati *et al.* [28] utilized over encryption to realize access control. Yu *et al.* [29] realized fine-grained access control in cloud computing by combining techniques of ABE, lazyrevocation and proxy re-encryption. Benaloh *et al.* [30] considered the security problem of Electronic Health Records (EHR). Narayan *et al.* [31] just combined bABE and PEKS [6] together to realize a patient-centric EHR management system. Li *et al.* [32] also tried to realize access control and keyword search over encrypted data by employing both ABE [18] and hybrid clouds.

Most of these works can be categorized into two groups, *key-based access control* (KBAC) and *attribute-based access control* (ABAC). KBAC [33], [34] usually assigns each file's decryption key directly to authorized users. When a user receives increasing number of such keys, its load on the keys management can be too high. To reduce the load, ABAC [25], [26], [29] attaches a set of attribute values to a user (resp. a file) and designs access policy for a file (resp. a user). A file can be accessed if and only if the attribute values satisfy the access policy. The access keys (e.g., the decryption keys in KBAC and the keys representing attribute values in ABAC) are usually required to be kept secretly to prevent data security from being compromised.

Therefore, the *conventional way* to perform encrypted search with access control is to conduct the search operations at the cloud server to take advantage of its large computation power, and leave the enforcement of access control at users' machines to keep their access keys from being disclosed. This separation of search and access control enforcement could lead to performance degradation, especially when users are assigned with different access permissions to search different encrypted cloud data. For example, in the conventional practice, a cloud server may perform search and transfer all the matching files to the users for them to decrypt the files. However, a user may not be allowed to access all the files and many of the transferred files have to be discarded, which leads to wasted network bandwidth and reduced service efficiency.

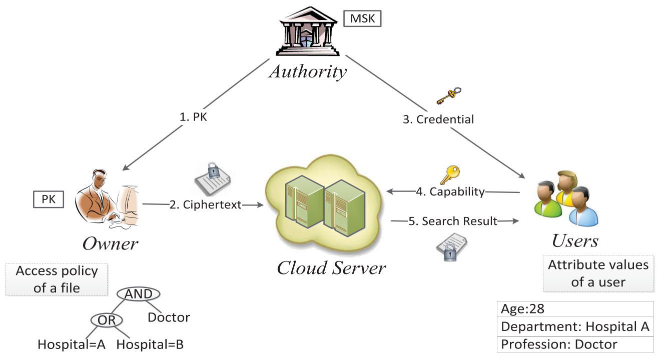


Fig. 1. The framework of KSAC. PK is the public keys, and MSK is the master secret key that should be securely kept. Credential is the set of keys standing for user's attribute values. Search capability is generated by using user's credential and his interested query.

C. Challenges to Be Addressed

Some attempts [30]–[32] have been made to integrate access control with keyword search over encrypted data, several challenges are still remained.

Challenge 1: For the sake of efficiency and convenience, multi-field search query and fine-grained access control must be supported. Moreover, to utilize cloud servers' computation power, we should place the enforcement of access control on servers while they are responsible for searching over encrypted data, so that a retrieved file not only matches the user's search query but also conforms to the user's access rights. In the process, we must ensure a user's access rights are not disclosed to any unauthorized parties.

Challenge 2: Many of proposed SE schemes [9], [19], [27], [31] require a role, such as data owner, to handle the search capability derivation for user's interested keywords *every time before search*. This requirement places heavy burden on data owners and significantly compromises the system scalability. The weakness should be mitigated by allowing user to locally derive the search capability.

D. Our Contributions

In this paper, we systematically study the issue of *Keyword Search with Access Control* (KSAC) over encrypted cloud data. Our **main contributions** are summarized as follows.

First, we propose a scalable framework as shown in Figure 1 that integrates multi-field keyword search with fine-grained access control. In the framework, every user authenticated by an authority obtains a set of keys called *credential* to represent his attribute values. Each file stored in the cloud is attached with an encrypted index to label the keywords and specify the access policy. Every user can use his credential and a search query to ***locally*** generate a *search capability*, and submit it to the cloud server who then performs search and access control. Finally, a user receives the data files that match his search query and allow his access. This design addresses the first challenge by fully leveraging the computation power of cloud server. It also solves the second challenge by dispersing the computation burden of capability generation to the users in the system.

Second, to enable such a framework, we make a novel use of Hierarchical Predicate Encryption (HPE) [35], to realize the derivation of search capability. Based on HPE, we propose our scheme named KSAC. It enables the service of both the keyword search and access control over multiple fields, and supports efficient update of access policy and keywords. KSAC also introduces some random values to enhance the protection of user's access privacy. To the best of our knowledge, KSAC is the ***first*** solution to simultaneously achieve the above goals.

Finally, we fully implement KSAC and conduct extensive evaluations to demonstrate its applicability.

II. Problem Formulation

A. System Model

In this paper, we consider a cloud-data-sharing system consisting of four entities, i.e., data owners, authority, data users and cloud server (shown in Figure 1). *Data owners* create data files, design the encrypted indices containing both keywords and access policy for each file, and upload the encrypted files along with the indices to the cloud server (step 2 in Figure 1). *Authority* is responsible to authenticate user's identity. It issues a set of keys as a credential to represent user's attribute values (step 3). *Data user* generates a search capability according to his credential and a search query, and submits it to the cloud server for file retrieval (step 4). The *cloud server* stores the encrypted data and performs search when receiving search capabilities from users (step 5).

B. Threat Model

Like many previous SE schemes [1], [9]–[11], [19], [36], [37], the cloud server is assumed to be "honest but curious". It means that the server will honestly execute the pre-defined protocol, but it is also "curious" to learn the information in terms of index, user's query and attribute values. Besides, the authority is assumed to be honest to generate user's credential according to his possessed attribute values.

C. Design Goals

- **Data Confidentiality and Index Privacy:** The data confidentiality should be protected against the cloud server and unauthorized users, so that the file content can be seen by authorized users only. Index privacy indicates the cloud server should be unaware of the attribute values in access policy and the keywords embedded in the index.
- **Fine-grained Access Control and Multi-Field Keyword Search:** The system should support fine-grained access policy and multi-field keyword search. In this paper, we mainly consider the access policy and the search query in *Conjunctive Normal Form* (CNF) over multiple fields.
- **Efficiency:** The system should promise the efficiency for general operations in practical environment, such as search and search capability derivation.
- **Adaption to Frequent Updates:** To cope with the scenario with frequent updates, either to access policy or to keywords, the system should provide an efficient update strategy.

D. Notations and Definitions

We first list their definitions and explanations in Table I.

TABLE I
Some Frequently Used Symbols and Descriptions

Symbols	Descriptions
Defined in Section II	
$\mathcal{S}, \vec{S}$	access policy, access policy vector (APV)
$\mathcal{W}, \vec{W}$	keywords of the file, keyword vector (KV)
W_i, w_i	i-th keyword field, keyword of W_i
d	number of keyword fields
e	number of attribute fields
$\mathcal{A}, \vec{A}$	attribute values, attribute vector (AV)
A_i,a_i	i-th attribute field, attribute value of A_i
$\langle \vec{x}, \vec{y} \rangle$	inner-product of vectors $\vec{x}$ and $\vec{y}$
Defined in Section III	
n_1	length of vector space for access control in KSAC
n_2	length of vector space for keyword search in KSAC
$\mathcal{D}_{\vec{A}}$	credential originated from $\vec{A}$
$\mathcal{Q}, \vec{Q}$	user's query, query vector (QV)
$T_{\vec{A},\vec{Q}}$	search capability generated from $\mathcal{D}_{\vec{A}}$ and $\vec{Q}$
$\vec{a}\|\|\vec{b}$	vector produced by the concatenation of $\vec{a}$ and $\vec{b}$
$\mathbb{F}_q$	the finite field

1) Attribute: The "*attribute*" is used to characterize the identity of a user. A user's attribute values are denoted as $\mathcal{A} := (A_1 = a_1, \cdots, A_e = a_e)$, where A_i is the i-th attribute field and a_i is the attribute value of A_i. e denotes the number of attribute fields. For example, a user's attribute values are "`Age`=37, `Profession`=Professor", then "37" (i.e., a_i) is an attribute value of attribute "`Age`" (i.e., A_i).

2) Predicate: In this paper, we use the predicate vector $\vec{v}$ to express the search query and access policy, and use value vector [1] $\vec{x}$ to express the keywords and user's attribute values. In KSAC, the decryption can succeed if and only if the inner product $\langle \vec{x}, \vec{v} \rangle = 0$. The transformation from the query and values into the vector form can be referred below.

3) Transformation to Vector Form: According to [27], one can represent a logical formula in the vector form. Suppose F_i denotes the i-th field (e.g., the keyword field or the attribute field), then a general CNF formula over s fields can be represented as $\mathcal{F} := (F_1 \in (f_{1,1}, \cdots, f_{1,m_1})) \wedge (F_2 \in (f_{2,1}, \cdots, f_{2,m_2})) \wedge \cdots \wedge (F_s \in (f_{s,1}, \cdots, f_{s,m_s}))$, where $f_{i,j}$ is the j-th requested value over F_i. It can be converted into the polynomial form with s univariate polynomials as: $f(F_1, F_2, \cdots, F_s) := r_1(F_1 - f_{1,1}) \cdots (F_1 - f_{1,m_1}) + r_2(F_2 - f_{2,1}) \cdots (F_2 - f_{2,m_2}) + \cdots + r_s(F_s - f_{s,1}) \cdots (F_s - f_{s,m_s})$, where r_i $(1 \le i \le s)$ is a randomly selected value. The *predicate vector* of the formula $\mathcal{F}$ can be derived as $\vec{p} := (p_{1,m_1}, \cdots, p_{1,1}, \cdots, p_{s,m_s}, \cdots, p_{s,1}, p_0)$, where $p_0 := \sum_{i=1}^{s}((-1)^{m_i} \cdot r_i \cdot \prod_{j=1}^{m_i} f_{i,j})$ and $p_{i,j}$ denotes the coefficient of F_i^j in $f(F_1, \cdots, F_s)$. For a value set $\mathcal{V} := (F_1 = v_1, \cdots, F_s = v_s)$, it can be converted into a *value vector* $\vec{v} := (v_1^{m_1}, \cdots, v_1, \cdots, v_s^{m_s}, \cdots, v_s, 1)$.

So that if $\mathcal{V}$ satisfies $\mathcal{F}$, then the equation $\langle \vec{v}, \vec{p} \rangle = f(v_1, \cdots, v_s) = 0$ establishes with overwhelming probability.

[1]In some previous works [27], it is called *attribute vector*. To distinguish it with the term "attribute", we denote it as the value vector in this paper.

Moreover, the length of $\vec{v}$ and $\vec{p}$ is $\sum_{i=1}^{s} m_i + 1$, which is linear with the total number of searched keywords (i.e., $\sum_{i=1}^{s} m_i$).

4) Access Policy Vectors and Attribute Vectors: An *access policy* is a unique logical expression over attribute values to specify the authorized users. For example, an access policy "Profession= student $\wedge$ Department=chemistry" only grants the student of chemistry department the access right. The access policy can be transformed into the predicate vector called *access policy vector* (APV) by following the transformation referred above. The *attribute vector* (AV) is the value vector transformed from the user's attribute values.

5) Keyword and Search Query: The *keywords* refer to the specific terms that characterize file content. For a file, its keyword set can be expressed as $\mathcal{W} := (W_1 = w_1, \cdots, W_d = w_d)$, where W_i is the i-th keyword field and w_i is the keyword of W_i. For example, the keywords of an Electronic Health Records (EHR) may be "Name=Alice, Age=30, illness=headache". Then "illness" is a keyword filed (i.e, W_i) and "headache" is a keyword (i.e., w_i) over this field.

A "*search query*" represents the user's interest, and is expressed in CNF [1], [38]–[40]. The search query and keywords can be transformed into the predicate vector called *query vector* (QV) and the value vector called *keyword vector* (KV) respectively.

6) Search Capability: A *search capability* includes the information of the search query and user's attribute values. For instance, a user's attribute values are "Age=37, Profession=professor" and the query is "Date=2013/1/1$\wedge$ Topic=meeting", then the search capability includes the logical information of "Age=27, Profession=professor, Date=2013/1/1 $\wedge$ Topic=meeting".

III. System Description

In this section, we will describe KSAC. Its **main ideas** are as follows. First, we can use vectors to represent the values (e.g., keywords and attribute values) and the CNF expression (e.g., access policy and search query). Second, we can utilize the delegation process in HPE to realize the derivation of search capability. Third, the encrypted index includes two components to respectively serve the decryption requests from the search capability and the credential.

A. An Introduction to HPE

Hierarchical predicate encryption (HPE) [35] is an cryptographic primitive that supports delegation of *predicate encryption* (PE) [27]. In HPE, a secret key $sk_{\vec{p}_l}$ for a predicate vector $\vec{p}_l$ can decrypt the ciphertext that associates with a value vector $\vec{v}$ if their inner-product $\langle \vec{p}_l, \vec{v} \rangle = 0$. In the delegation of HPE, for a vector $\vec{p}_{l+1} \notin span < \vec{p}_l >$, a more restrictive secret key $sk_{\vec{p}_l, \vec{p}_{l+1}}$ can be generated with the $sk_{\vec{p}_l}$ and $\vec{p}_{l+1}$ taken as input. $sk_{\vec{p}_l, \vec{p}_{l+1}}$ can decrypt a ciphertext tied with the value vector $\vec{v}$ if $\langle \vec{p}_l, \vec{v} \rangle = \langle \vec{p}_{l+1}, \vec{v} \rangle = 0$.

B. The Design of Index Format

According to the property of HPE, we give a novel design of the encrypted index as shown in Figure 2. The index information includes the specified access policy, the representative

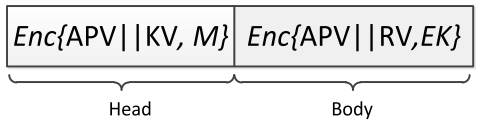


Fig. 2. The format of an encrypted index. For example, for **KSAC.BuildIndex** in Algorithm 1, M is set as $I_{\mathbb{G}_T}$ (i.e., the identity element of the group $\mathbb{G}_T$ used in HPE and is public.)

keywords, and the symmetric keys used to encrypt the file content. To build an encrypted index, the data owner first produces the "*body*" component to lock the symmetric key EK by utilizing the *access policy vector* (APV) and a *random vector* (RV). EK is used as the symmetric key to encrypt and decrypt the file content. This body component ensures that only the users whose attribute values satisfy the access policy can recover EK for file decryption. The recovery of EK performed by the unauthorized user will be rejected by APV and the cloud server's attempt to obtain EK by stealthily using the authorized user's search capability will be refused by RV.

The data owner further encrypts M, the representative keywords and the access policy, and produces the "*head*" component. This design ensures that the file can be retrieved only when the keywords match the query and the user's attribute values satisfy the access policy.

We now give a detailed procedure about how to construct the head component when given the keywords and the access policy. The construction of body component is similar. For the keywords $\mathcal{W} = (W_1 = w_1, \cdots, W_d = w_d)$ where d is the number of keyword fields, we can convert it into the keyword vector $\vec{W}$ according to the transformation referred in Section II-D. Suppose the access policy is $\mathcal{S} = (A_1 = a_{1,1} \vee \cdots \vee a_{1,p_1}) \wedge \cdots \wedge (A_e = a_{e,1} \vee \cdots \vee a_{e,p_e})$, where $a_{i,j}$ $(1 \le i \le e,\ 1 \le j \le p_i)$ denotes the j-th required attribute value in the field A_i, e is the number of attribute fields, and p_j represents the total number of required attribute values over the j-th attribute field. We can transform $\mathcal{S}$ into the *access policy vector* (APV) $\vec{S}$ according to the transformation referred in Section II-D. Then we concatenate these two vectors and output $\vec{S}||\vec{W}$. Finally, we call HPE.Enc to encrypt $\vec{S}||\vec{W}$ and M, and produce the head component.

Figure 2 shows that the encrypted index in KSAC contains the keywords as well as the access policies. When performing search operation, the cloud server first decrypts the head component to see if the decrypted result is M. If it is, then the file matches the capability. After obtaining the matching files, the user can decrypt the body component to free EK, which will then be used to decrypt the text information of the matching files.

C. System Description

Our proposed KSAC scheme is shown in Algorithm 1, which contains the following main algorithms.

- **KSAC.Setup**. This algorithm is operated by the authority to generate the public key and the master secret key. The authority chooses a security parameter λ and a format of hierarchy $\vec{\mu}$, invokes HPE.Setup, and outputs a public key PK and a master secret key MSK. PK will be

Algorithm 1: The Design of KSAC

- **KSAC.Setup**$(1^\lambda, \vec{u})$: Invoke $\mathsf{HPE.Setup}(1^\lambda, \vec{u})$ and output $PK := (1^\lambda, param, \mathbb{B})$ and $MSK := (X, \mathbb{B}^*)$.
- **KSAC.BuildIndex.**$(PK, \mathcal{S}, \mathcal{W}, I_{\mathbb{G}_T}, EK)$: Convert the access policy $\mathcal{S}$ and the keywords $\mathcal{W}$ into the access policy vector $\vec{S}$ and the keyword vector $\vec{W}$ respectively, choose a random vector $\vec{R}$, and compute $C^{head} := (\mathbf{C}_1^{head}, C_2^{head})$:=$\mathsf{HPE.Enc}(PK, \vec{S}||\vec{W}, I_{\mathbb{G}_T})$ and $C^{body} := (\mathbf{C}_1^{body}, C_2^{body})$:=$\mathsf{HPE.Enc}(PK, \vec{S}||\vec{R}, EK)$.
- **KSAC.GenCre**$(PK, MSK, \mathcal{A})$: Convert $\mathcal{A}$ into the attribute vector $\vec{A} := (a_1, \cdots, a_{n_1})$, and generate the credential $D_{\vec{A}} := (\mathbf{k}_{1,0}^*, \mathbf{k}_{1,1}^*, \mathbf{k}_{1,2}^*, \mathbf{k}_{1,n_1+1}^*, \cdots, \mathbf{k}_{1,n_1+n_2}^*)$:=$\mathsf{HPE.GenKey}(PK, MSK, \vec{A})$.
- **KSAC.DeriCap**$(\mathcal{D}_{\vec{A}}, \mathcal{Q}, PK)$: Transform the query $\mathcal{Q}$ into the query vector $\vec{Q} := (q_1, \cdots, q_{n_2})$, obtain $\mathbf{k}_{2,0}^*$ by partially calling $\mathsf{HPE.Delegate}_1(PK, D_{\vec{A}}, \vec{Q})$, and return $T_{\vec{A},\vec{Q}} := \mathbf{k}_{2,0}^*$.
- **KSAC.Search**$(T_{\vec{A},\vec{Q}}, C^{head}, PK)$: Run $\mathsf{HPE.Dec}(C^{head}, T_{\vec{A},\vec{Q}}, PK)$ and output True if the result is $I_{\mathbb{G}_T}$.
- **KSAC.RelEK**$(C^{body}, \mathcal{D}_{\vec{A}}, PK)$:Release the symmetric key EK by invoking $\mathsf{HPE.Dec}(C^{body}, \mathcal{D}_{\vec{A}}, PK)$.

published while MSK will be kept as a secret. The format of hierarchy $\vec{u}$ has two vector spaces, where n_i is the length of i-th vector space ($i := 1, 2$). In KSAC, the first vector space is defined as the attribute space, while the second space is the keyword space.

- **KSAC.BuildIndex**. This algorithm is performed by data owner to produce the encrypted searchable index. The owner first converts the access policy and keywords into the *access policy vector* (APV) and *keyword vector* (KV), and chooses a *random vector* (RV). To generate the head component C^{head}, the owner calls $\mathsf{HPE.Enc}$ to encrypt $I_{\mathbb{G}_T}$ (i.e., the identity element of group $\mathbb{G}_T$ that used in HPE) with the concatenation of APV and KV. To generate the body component, the owner calls $\mathsf{HPE.Enc}$ and encrypts the symmetric key EK with the concatenation of APV and RV, where EK is used for file encyption/decryption.
- **KSAC.GenCre**. This algorithm is run by the authority to generate the credential based on user's attribute values. According to user's attribute values $\mathcal{A}$, the authority first converts $\mathcal{A}$ into the *attribute vector* (AV) and then issues the corresponding *credential* $D_{\vec{A}} := (\mathbf{k}_{1,0}^*, \mathbf{k}_{1,1}^*, \mathbf{k}_{1,2}^*, \mathbf{k}_{1,n_1+1}^*, \cdots, \mathbf{k}_{1,n_1+n_2}^*)$ by calling $\mathsf{HPE.GenKey}$. In $D_{\vec{A}}$, $\mathbf{k}_{1,0}^*$ is used as the decryption key to release EK from the body component of the encrypted index for file decryption, $(\mathbf{k}_{1,1}^*, \mathbf{k}_{1,2}^*)$ are used for re-randomization in the search capability derivation, and $(\mathbf{k}_{1,n_1+1}^*, \cdots, \mathbf{k}_{1,n_1+n_2}^*)$ are used for the derivation of search capability [35].
- **KSAC.DeriCap**. This algorithm is executed by data user to derive the search capability according to his query and credential. For a query $\mathcal{Q}$, the user transforms it into the *query vector* (QV) and produces the *search capability* $T_{\vec{A},\vec{Q}}$. Notice that the user does not need to complete the algorithm $\mathsf{HPE.Delegate}_1$, but just computes $\mathbf{k}_{2,0}^*$ to make it serve as the capability. $\mathbf{k}_{2,0}^*$ can be treated as the decryption key to decrypt the head component of the encrypted index (The description of $\mathsf{HPE.Delegate}$ can be referred to [35].).
- **KSAC.Search**. This algorithm is performed by the cloud server. After receiving the capability $T_{\vec{A},\vec{Q}}$, the cloud server calls $\mathsf{HPE.Dec}$. If the result is $I_{\mathbb{G}_T}$ that is public, then it indicates the condition $\langle \vec{Q}, \vec{W} \rangle = \langle \vec{A}, \vec{S} \rangle = 0$ establishes, meaning that the file's keywords match the search query (i.e., $\langle \vec{Q}, \vec{W} \rangle = 0$) and the user is allowed to access the file (i.e., $\langle \vec{A}, \vec{S} \rangle = 0$). If a mismatch happens, the cloud server cannot determine whether the keywords mismatch the search query or the files refuse the user's access. Finally, the cloud server returns the ciphertext along with the body component C^{body} to the user.
- **KSAC.RelEK**. This algorithm is executed by the user to decrypt the file. After obtaining matching files, the authorized user can successfully release EK to decrypt the encrypted file content, since $\langle \vec{A}, \vec{S} \rangle = \langle \vec{0}, \vec{R} \rangle = 0$ holds.

An Example: Alice (i.e., data owner) wishes to share her health records with her personal doctor Bob (i.e., data user) in hospital A, then she can combine the access policy $\mathcal{S}$ "`Hospital`=A $\wedge$ `Name`=Bob" with keywords of her records before uploading them to the cloud. Bob, who keeps the credential standing for the attribute values "`Hospital`=A, `Name`=Bob" distributed by the hospital A (i.e., authority), can generate his query like "`Name`=Alice $\wedge$ `Date`=4/20/2014" to timely learn Alice's body status. If Carl is a new doctor for Alice in hospital A, to grant him the access permission, Alice can change the access policy to "`Hospital`=A $\wedge$ (`Name`=Carl $\vee$ `Name`=Bob)". This procedure relates to the access policy update referred in Section III-D.

D. Access Policy Update and Keyword Update

To adapt to the scenario with frequent updates, such as the update to the access policy, KSAC should efficiently support the access policy update as well as keyword update. Suppose the owner plans to update the access policy vector (APV) from $\vec{S} = (s_1, \cdots, s_{n_1})$ to $\vec{S}' = (s_1', \cdots, s_{n_1}')$ where n_1 is the length of APV, then the head and the body of the index should be correspondingly renewed. A straightforward way of updating C^{head} and C^{body} by directly running $\mathsf{HPE.Enc}(PK, \vec{S}'||\vec{W}, I_{\mathbb{G}_T})$ and $\mathsf{HPE.Enc}(PK, \vec{S}'||\vec{R}, EK)$ will pay for the re-encryption cost for the unchanged $\vec{W}$ and $\vec{R}$ for every update, causing considerable waste of computation capacity.

```
Algorithm 2: Access Policy Update
1) keep lists KVCL (i.e., $\{C^{\vec{W}}\}$) and RVCL (i.e., $\{EK, C^{\vec{R}}\}$), where $C^{\vec{W}} := (\mathbf{C}_1^{\vec{W}}, C_2^{\vec{W}})$ :=HPC.Enc($PK, \vec{W}, I_{\mathbb{G}_T}$), $C^{\vec{R}} := (\mathbf{C}_1^{\vec{R}}, C_2^{\vec{R}})$ :=HPC.Enc($PK, \vec{R}, I_{\mathbb{G}_T}$).
2) convert the updated access policy $\mathcal{S}'$ into $\vec{S}'$, and produce $C^{\vec{S}'} := (\mathbf{C}_1^{\vec{S}'}, C_2^{\vec{S}'})$ :=HPE.Enc($PK, \vec{S}', I_{\mathbb{G}_T}$).
3) uniformly pick $\delta_1$ from $\mathbb{F}_q$, and update the head component, where $\mathbf{C}_1^{head'} := \mathbf{C}_1^{\vec{S}'} + \delta_1 \cdot \mathbf{C}_1^{\vec{W}}$, $C_2^{head'} := C_2^{\vec{S}'} \cdot (C_2^{\vec{W}})^{\delta_1}$
4) uniformly pick $\delta_2$ from $\mathbb{F}_q$, and update the body component, where $\mathbf{C}_1^{body'} := \mathbf{C}_1^{\vec{S}'} + \delta_2 \cdot \mathbf{C}_1^{\vec{R}}$, $C_2^{body'} := C_2^{\vec{S}'} \cdot (C_2^{\vec{R}})^{\delta_2} \cdot EK$
5) upload $C^{head'} := (\mathbf{C}_1^{head'}, C_2^{head'})$ and $C^{body'} := (\mathbf{C}_1^{body'}, C_2^{body'})$ to the cloud.
```

Here, we make the following changes to efficiently update the access policy without loss of security (shown in Algorithm 2): when creating C^{head}, the owner should locally keep a list named *keyword vector component list* (KVCL) (step 1). Every item in KVCL is formed by $C^{\vec{W}}$ produced by calling HPE.Enc($PK, \vec{W}, I_{\mathbb{G}_T}$). When the owner renews the access policy from $\mathcal{S}$ to $\mathcal{S}'$, he can convert $\mathcal{S}'$ into $\vec{S}'$ first and then calculate $C^{\vec{S}'}$ (step 2). To efficiently update the encrypted index, the owner uniformly choose a value δ_1 from $\mathbb{F}_q$ ($\mathbb{F}_q$ is a finite field) to randomize the distribution of $C^{\vec{W}}$ and compute the updated head component $C^{head'}$ (step 3).

Similarly, when the owner plans to update C^{body}, he should keep a *random vector component list* (RVCL), in which every item is formed by the symmetric key EK and $C^{\vec{R}}$ produced by calling HPE.Enc($PK, \vec{R}, I_{\mathbb{G}_T}$) (see step 1). When the APV changes from $\vec{S}$ to $\vec{S}'$, the owner produces the updated body component $C^{body'}$ by reusing the item in RVCL (step 4).

Lastly, the owner requires the cloud server to replace the expired index with the updated index (step 5). It is intuitive that keeping KVCL and RVCL improves the efficiency of access policy update by sacrificing a small amount of storage space. The workflow of keyword update is similar, and one difference is that only the head component is required to be updated in keyword update.

E. The Protection of Access Privilege Privacy

In the above protocols, the cloud server can still get a rough understanding about a user's access privilege with the number of searches increases, just by listing the files ever returned to that user. To enhance access privacy, we inject the *noise* which is actually a random number into the search capability and the index, so that some files exceeding the user's privilege might be returned and user's actual access privilege will be concealed against the cloud server.

When building an index, the owner firstly chooses a random number o_j and designs the new access policy as $\mathcal{S}' := \mathcal{S} \vee o_j$. A user can produce the search capability based on the search query and another noise o_i, so that even his access is refused by $\mathcal{S}$, he can still receive the file if $o_i = o_j$ and the keywords match the query. Thus, the cloud server will be misled under the effect of the noise. Even the files that are not covered by the user's access privilege may be returned (i.e., in the case when o_j embedded in the encrypted index equals o_i in the capability, but the index does not match the capability), the user cannot obtain the symmetric keys to decrypt them due to the restriction of $\mathcal{S}$ in C^{body} (described in the algorithm KSAC.RelEK in Section III-C). This method not only respects the access policy, but also defends the user's privilege privacy.

F. Analysis of KSAC

1) Data Confidentiality and Index Privacy: In the design of KSAC, we employ HPE [35] as the *black box* and the protocols of KSAC can be mapped to the security game of HPE. Therefore, KSAC has the same formal security definition and security proofs as HPE, i.e., KSAC reaches the selectively attribute-hiding security against chosen plaintext attack under the RDSP and IDSP assumptions [35]. It means that the advantage is negligible for a computationally bounded adversary (e.g., the cloud server) to distinguish the two cipehrtexts of two encrypted vectors without a capability to differentiate the two ciphertexts.

The C^{head} and C^{body} in KSAC can be treated as the ciphertext of encrypted vectors (i.e., $\vec{S}||\vec{W}$ and $\vec{S}||\vec{R}$) in HPE, while the search capability $T_{\vec{A},\vec{Q}}$ in KSAC can be regarded as the capability in HPE. Therefore, being treated as the adversary in the threat model of HPE, [2] the cloud server can only obtain the ciphertext and the capability, where the distribution of ciphertext in KSAC are consistent with that in HPE. This indicates that, in the security game defined in HPE [35], the cloud server not only cannot obtain any useful information about *EK* (i.e., the message in HPE), but also cannot learn the access policy vector $\vec{S}$ and keyword vector $\vec{W}$. Thus KSAC can achieve the selectively attribute-hiding security as HPE to guarantee index privacy.

In KSAC, the file content is first encrypted by the symmetric key EK and EK is then encrypted by HPE. Thus, the data confidentiality is ensured by the symmetric algorithm and HPE together.

2) Fine-Grained Access Control and Multi-Field Search Query: KSAC uses the access policy vector (APV) to represent the access policy specified in CNF formula over multiple

[2]The adversary defined in the threat model of HPE will try to learn the encrypted information, which comforts to the "curious" setting of the cloud server in KSAC.

attribute fields. Meanwhile, KSAC also allows users to employ a query vector to express the search query represented in CNF among multiple keyword fields.

3) Access Privilege Privacy Analysis: Suppose the actual access permission of a user without the noise disturbance is $\mathbb{P}$, we use $|\mathbb{P}|$ to denote the number of files in $\mathbb{P}$. The returned files purely caused by the effect of noise (i.e., the returned files that the user is not permitted to access) after t searches can be expressed as $\mathbb{N}_t = \bigcup_{i=1}^{t} \mathbb{D}_i$, where $\mathbb{D}_i$ is composed of the newly returned files that purely caused by the noise in the i-th search and have not appeared in the previous $(i-1)$ searches before. This definition means that the disturbed file will just be recorded the first time it is retrieved.

Then the number of files that are returned by the effect of noise is $|\mathbb{N}_t| = \sum_{i=1}^{t} |\mathbb{D}_i|$, where $|\mathbb{D}_i|$ denotes the number of files in $\mathbb{D}_i$, thus the concept of *disturbance rate* after t searches can be given by

$$R_t = \frac{|\mathbb{N}_t|}{|\mathbb{P}|} \tag{1}$$

The disturbance rate R_t represents the protection degree of the user's access permission against the cloud server after t searches. The larger disturbance rate usually achieves the stronger protection for user's access privilege. Moreover, it is easy to observe that the disturbance rate will increase as the number of disturbed files becomes larger.

4) Communication Cost Introduced by Noise: Without loss of generality, in the i-th search, we assume that for a noise value o_i in the capability, the probability of a file that is tied with this noise is P_{o_i}. For a search, suppose the probability for a file to satisfy the search condition (i.e., the keywords of a file match the query and user's attribute values satisfy the access policy) is P_{q_i}. We assume that these two kinds of probability are independent, then a file is returned *purely* due to the noise is $P_{o_i}(1-P_{q_i})$. If we let the total number of files be $|\mathbb{D}|$, then the number of extra returned files caused by the noise o_i can be formulated as Equation (2), while the number of real retrieved files N_{q_i} caused by the effect of the search condition can be expressed as Equation (3).

$$N_{o_i} = |\mathbb{D}| \cdot P_{o_i} \cdot (1 - P_{q_i}) \tag{2}$$

$$N_{q_i} = |\mathbb{D}| \cdot P_{q_i} \tag{3}$$

Suppose a user launches t searches, without loss of generality, we assume that the first r searches $(r \le t)$ include the noise values $\{o_i\}_{i=1}^{r}$, then the ratio λ between extra communication cost and the real communication cost after t searches can be formulated as follows:

$$\lambda = \frac{\sum_{i=1}^{r} N_{o_i}}{\sum_{i=1}^{t} N_{q_i}} = \frac{\sum_{i=1}^{r} P_{o_i}(1-P_{q_i})}{\sum_{i=1}^{t} P_{q_i}} \tag{4}$$

Equation (4) shows that the value r and the probability of P_{o_i} can be adjusted to ensure that the introduction of the noise value only incurs acceptable extra cost to users.

5) Security Analysis of Access Policy Update: When updating the access policy of a file from $\vec{S}$ to $\vec{S}'$, on one hand, the data owner first chooses the coefficients uniformly to encrypt $\vec{S}'$, thus the coefficients of $\mathbf{b}_i$ and $\mathbf{d}_{n_1+n_2+1}$ in $C^{\vec{S}'}$ are uniformly distributed among $\mathbb{F}_q$, for $i = 1, \cdots, n_1, n_1+n_2+3$. $\mathbf{b}_i$ $(i = 1, \cdots, n_1, n_1+n_2+3)$ and $\mathbf{d}_{n_1+n_2+1}$ are the bases in PK and employed to encrypt a targeted vector in HPE. More detail about the bases of HPE can be referred to [35].

On the other hand, the owner then fetches the pre-stored items $C^{\vec{W}}$ and $C^{\vec{R}}$, and uses the uniformly selected values δ_1 and δ_2 to ensure that the coefficients of $\mathbf{b}_i$ and $\mathbf{d}_{n_1+n_2+1}$ in $\delta_1 \cdot C^{\vec{W}}$ and $\delta_2 \cdot C^{\vec{R}}$ are uniformly distributed among $\mathbb{F}_q$ (for $i = n_1+1, \cdots, n_1+n_2, n_1+n_2+3$) and the coefficient of $\mathbf{d}_{n_1+n_2+1}$ equals the exponent of g_T, where g_T is the generator of the group $\mathbb{G}_T$ in HPE.

Therefore, the generation of $C^{head'}$ will promise that the distributions of $C^{head'}$ is consistent with that of directly calling $\mathsf{HPE.Enc}(PK, \vec{S}'||\vec{W}, I_{\mathbb{G}_T})$ in the cloud server's view. This guarantee also establishes when generating $C^{body'}$. Therefore our proposed access policy update can achieve the same security strength with that of direct calling HPE algorithms.

Based on the same principle, the keyword update is secure as directly invoking $\mathsf{HPE.Enc}(PK, \vec{S}||\vec{W}', I_{\mathbb{G}_T})$.

G. Complexity Analysis of KSAC

Suppose n_1 (resp. n_2) denotes the length of the vector space for access control (resp. keyword search). From the construction of HPE [35], the length of vector $\mathbf{b}$ and $\mathbf{b}^*$ is $O(n_1+n_2)$. Therefore, the computation complexity for index encryption is $O((n_1+n_2)^2)$. The per-credential generation complexity is $O((n_1+n_2)^2)$. As KSAC does not require to fully execute $\mathsf{HPE.Delegate}_1$, but just produces $\mathbf{k}^*_{2,0}$, the per-capability derivation complexity is $O(n_2(n_1+n_2))$. Finally, the search operation needs to perform the pairing operation to each pair of elements of the vectors embedded in the index and capability, thus the computation complexity will be $O(n_1+n_2)$.

IV. Performance Evaluation

We fully realize KSAC based on Java Pairing-Based Cryptography (JPBC) Library [41]. We first choose two real data sets "Nursery Data Set" [42] and "Adult Data Set" [43] to serve as the shared files and characterize data users. Second, we try to make the number of shared data files approach or even more than the average number of files shared by per user in real cloud storage systems. Specifically, the number of instances in "Nursery Data Set" is 12,960, which is much more than the average number of shared files per user reported in two typical cloud storage systems (i.e., 55 per Box user [44]). Our server is equipped with 2.10GHZ Intel® Core 2 Duo CPU and 4GB RAM. The operating system running on the server is Ubuntu (version: 11.04) and the kernel is Linux Ubuntu 2.6.38-8-generic.

We compare KSAC with another system that is composed of *Predicate Encryption* (PE) [27] and MRQED [9], and we call it "P+M". P+M utilizes PE for access control and employs MRQED for keyword search. The three balance primes in PE are selected with 342 bits each to equivalently achieve the same security strength provided by the group in KSAC whose order is 160-bits. Moreover, we also assume $d = 1$ in MRQED.

We define a format of hierarchy $\vec{u}$ with 2 levels. The first level is a vector space for access control (e.g., APV/AV) whose

TABLE II
STORAGE OVERHEAD UNDER REAL DATA SETS

	PK/MSK	Credential	Index	Capability
KSAC (KB)	143.4/191.3	94.5	***3.1***	***3.0***
P+M [27] [9] (KB)	18.6/18.5	0	11.5	13.0

length is n_1, and the second level is a query vector space (e.g., QV/KV) with the length n_2. We select attributes "*Sex*" and "*Occupation*" with 2 values and 14 values respectively from the "Adult Data Set" [43] to serve as user's attribute values. According to the transformation in Section II, the length of the first vector space is $n_1 := 2 + 14 + 1 := 17$. We then choose the "Nursery Data Set" [42] to denote data files, where attributes and corresponding values in this data set are treated as keyword fields and keywords respectively in our evaluation. There are 8 fields in the data set, among which there are 4 fields that have 3 possible keywords, 2 fields that have 4 possible keywords, 1 field that has 2 possible keywords, and 1 field that has 5 possible keywords. Therefore, $n_2 = 4 \times 3 + 2 \times 4 + 1 \times 2 + 1 \times 5 + 1 = 28$.

A. Experiment Setup

We adopt type A elliptic parameter [41] and select the group order as 160 bits to provide 1024 bits discrete log security strength. From the aspect of storage overhead, an element takes up 65 Bytes in compressed form in both KSAC and MRQED if the size of the base field for the group is 512 bits when q is 20 B. When $n_1 = 17$, $n_2 = 28$, then it needs 143.4 KB and 191.3 KB for KSAC to keep PK and MSK.

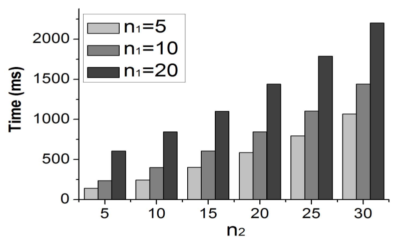
We also evaluate the storage cost of the comparison system P+M. For PE, an element in the composite group $\mathbb{G}$ can be represented by 130 Bytes and it needs 43 Bytes to represent the order of the subgroups. If $n_1 = 17$, then it needs 4.8 KB and 4.7 KB to keep PK and MSK in PE, respectively. For MRQED, the size of PK and MSK is the same, i.e., 13.8 KB when $n_2 = 28$. Thus the PK and MSK in the system P+M are 18.6 KB and 18.5 KB. The comparison on the storage overhead is shown in Table II. Though it requires a little more storage space to maintain the PK and MSK in KSAC when compared with P+M, it is less of an issue today because of the lower price of the storage space.

B. Experiment Results

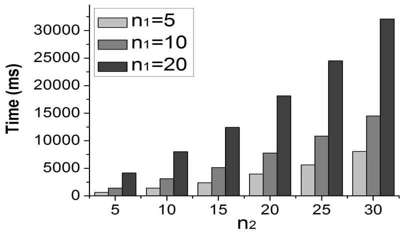
Experiment 1 (Index Building): For index construction, we mainly consider the time to create C^{head} and C^{body} and show the averaged result in Figure 3(a). One can observe that the time scales as $O(n_2^2)$ when n_1 is fixed. The size of C^{head} is the same to that of C^{body}, and when $n_1 = 17$, $n_2 = 28$, the creation time and the size of head/body part are 1.74 sec and 3.1 KB respectively. [1]

For P+M, to encrypt the access policy, it takes 0.42 sec and 4.6 KB for PE to generate the encrypted index and keep it when $n_1 = 17$. To encrypt the keywords, the per-index encryption time in MRQED is 1.28 sec when $n_2 = 28$ and the storage cost is 6.9 KB. Therefore, the total size of an encrypted index in P+M is 11.5 KB and the per-index encryption time is 5.11 sec. The results are listed in Table II and Table III.

[1]This value is obtained in our evaluation. Figures in this experiment are to show the used time when the values of n_1 and n_2 change respectively. Due to page limit, it is difficult to illustrate the time for all the pairs of (n_1, n_2).



(a)



(b)

Fig. 3. Comparison on encryption time and credential generation time. (a) Encryption time. (b) Credential generation time.

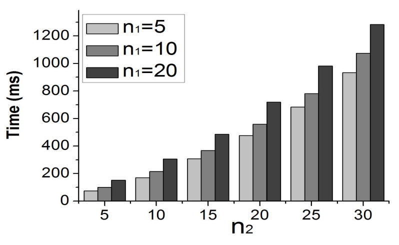
TABLE III
OPERATION TIME UNDER REAL DATA SETS

	BuildIndex	GenCre	DeriCap	Search
KSAC (Sec)	1.74	22.15	***1.08***	***0.12***
P+M [27] [9] (Sec)	1.70	0	5.11	1.69

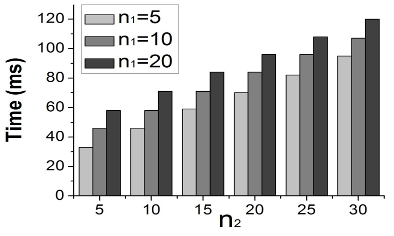
According to Table III, to build an encrypted index, KSAC needs a bit more time than P+M. However, building an index is still a *one-time operation* and is not very significant. In the environment with keyword search and access control, the most dominated operations are the search capability derivations and the search operations, which should be placed with special attention.

Experiment 2 (Credential Generation): For credential generation, we test 100 cases when n_1 and n_2 change and show the averaged results in Figure 3(b). To generate a valid credential, $O(n_2^2)$ power operations and $O(n_2^2)$ multiplication operations should be carried out when n_1 is fixed. Most of the credential generations are one-time cost, thus we argue that the cost is still reasonable. Some special hardware aiming at improving the power efficiency can also be adopted. For example, Elliptic Semiconductor CLP-17 [9] can decrease the time of exponential operation from 6.4 ms to 30 us. When $n_1 = 17$, $n_2 = 28$, the credential size and the generation time in KSAC are 94.5 KB and 22.15 sec, respectively.

Both PE and MRQED in P+M do not require users to keep any "credential-like" materials and demand them to request for the capability from the MSK keeper (either the data owner or the authority), therefore the needed storage space and the credential generation time is zero as shown in Table II and Table III respectively. This comparison also indicates that

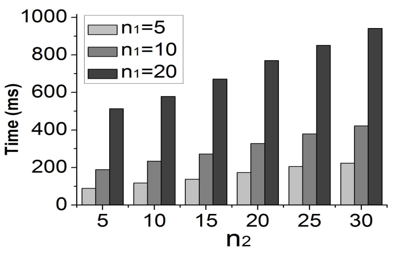


(a)

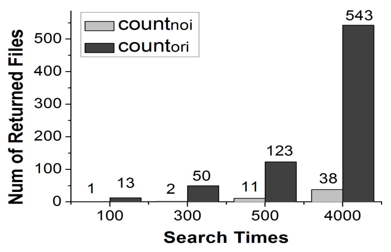


(b)

Fig. 4. Comparison on capability derivation time and search time. (a) Capability derivation time. (b) Search time.



(a)



(b)

Fig. 5. Comparison on access policy update time and number of returned files. (a) Access policy update time. (b) Number of returned files.

KSAC though increases a small number of storage capacities at the client machines, it grants data users the ability of capability derivation..

Experiment 3 (Search Capability Derivation): The evaluated results of search capability derivation are shown in Figure 4(a). We can observe that when $n_1 = 17$, $n_2 = 28$, deriving a search capability calls for about 1.08 sec, which is quite applicable in the real scenario. For the storage overhead, when $n_1 = 17$, $n_2 = 28$, it needs about 3.0 KB to store a valid search capability, which will not bring much storage burden to the user. The user can also keep the frequently used search capabilities to reduce the time cost in multiple search capability generations.

For the comparison system P+M, to enforce access control, the time of generating a secret key in PE is about 3.84 sec when $n_1 = 17$. Meanwhile, the keyword search functionality offered by MRQED will also require 1.27 sec when $n_2 = 28$. Therefore, the total needed time to derive a search capability in P+M is 5.11 sec as shown in Table III. In addition, PE system requires the MSK keeper (e.g., the data owner) to be responsible for the secret key generation. This setting will easily make the MSK keeper become the bottleneck of the system, once the access requests increase sharply. Table II also list the storage cost of a search capability in the system P+M when being evaluated by the selected two data sets.

From the evaluations above, *data user in KSAC has to afford most of the burden during the whole procedure of capability derivation, which makes KSAC more scalable than the comparison system P+M. Moreover, the capability derivation is more frequently executed compared with index building operations, and the efficiency of KSAC on this aspect also indicates its advantage when deployed in the real scenario.*

Experiment 4 (Search): Among all the operations in KSAC, search operation is generally the most frequently executed operation. It will be invoked whenever a user plans to retrieve his interested files and thus has a significant impact on user experience. Therefore, the search efficiency of KSAC is usually the greatest concern when being deployed in the real cloud scenario.

We present the average per-index search time when the length of n_1 and n_2 vary. The final results are shown in Figure 4(b). When $n_1 = 17$, $n_2 = 28$, it takes about 0.12 sec for KSAC to judge if a file satisfies the search capability. The search time of KSAC will be further decreased once the parallel technologies are introduced. Thus we believe that KSAC is acceptable for practical use in real cloud application.

We also evaluate P+M on the search efficiency. For PE, if $n_1 = 17$, the time to check user's access privilege after preprocessing is about 1.3 sec. For MQRED, it takes $O(n_2)$ pairing operations. When $n_2 = 28$, the search time is 0.39 sec. Therefore, for the system P+M that is based on PE [27] and MRQED [9], the search time in P+M should be 1.69 sec, which is 14.1 times than that in KSAC.

Experiment 5 (Access Policy Update): Moreover, we record the average time to generate a new access policy in Figure 5(a) and measure the storage overhead to maintain KVCL and RVCL. One can observe that the time to update the access policy is linear to n_1 when n_2 is fixed. If the owner updates the access policy of a file when $n_1 = 17$ and $n_2 = 28$, then he should pay 0.91 sec to renew the head/body component, and allocate merely 2.2 KB storage space for each item of KVCL/RVCL. Furthermore, the needed time to update the C^{head}(or C^{body}) in KSAC is almost half of that to directly re-produce C^{head}(or C^{body}), proving the efficiency of access policy update mechanism in KSAC.

Experiment 6 (Privacy Protection When Noise Is Introduced): In this test, we assume a data owner possesses 100 files, where each user is given two attribute values. For each file, we choose a set of keywords from Car Evaluation Database, design the access policy by choosing

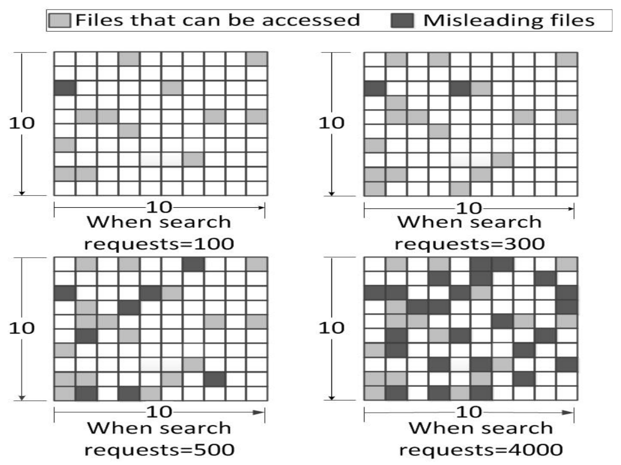


Fig. 6. The cloud server's vision about user's access pattern.

the appropriate attribute values, and assign it with a noise ranging from 0 to 400. A user that is allowed to access 17 files can generate a search capability derived from the random keywords every time. We accumulate the returned files when he issues 100, 300, 500 and 4,000 search requests respectively, and compare his real access privilege with the access pattern (files that are ever received by the user) in Figure 6. The access pattern will be far away from the user's real access privilege with the search requests increase, which does conceal user's access permission against the cloud server.

We also measure the cost brought by introducing the noise. We use count$_{ori}$ to denote the total number of the returned files which are originally allowed to be accessed by the user, and employ count$_{noi}$ to represent the number of the returned files that are beyond the user's access right. When the user issues 100, 300, 500 and 4,000 search requests respectively, the comparison in Figure 5(b) shows that the unneeded files will not take up much overhead (around 7%). Moreover, the user can further reduce the cost by adjusting the frequency of noise injection.

V. Conclusions

In this paper, we propose a scalable framework that allows users to locally derive the search capability by utilizing both their credentials and a search query. We then utilize HPE to realize this framework and present KSAC. KSAC realizes the fine-grained access control and multi-field keyword search, enables efficient update of both access policy and keywords, and protects user's access privacy. The results show that KSAC just needs 1.08 sec for per-capability generation, and takes 0.12 sec for match judgement between a search capability and an encrypted index.

References

[1] Z. Shen, J. Shu, and W. Xue, "Keyword search with access control over encrypted data in cloud computing," in *Proc. IEEE/ACM IWQoS*, May 2014, pp. 87–92.

[2] J. Shu, Z. Shen, and W. Xue, "Shield: A stackable secure storage system for file sharing in public storage," *J. Parallel Distrib. Comput.*, vol. 74, no. 9, pp. 2872–2883, Sep. 2014.

[3] M. Tinghuai *et al.*, "Social network and tag sources based augmenting collaborative recommender system," *IEICE Trans. Inf. Syst.*, vol. 98, no. 4, pp. 902–910, 2015.

[4] Y. Ren, J. Shen, J. Wang, J. Han, and S.-Y. Lee, "Mutual verifiable provable data auditing in public cloud storage," *J. Internet Technol.*, vol. 16, no. 2, pp. 317–323, 2015.

[5] J. Shu, Z. Shen, W. Xue, and Y. Fu, "Secure storage system and key technologies," in *Proc. 18th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2013, pp. 376–383.

[6] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. ACNS*, Jun. 2004, pp. 31–45.

[7] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security*. Berlin, Germany: Springer-Verlag, 2005.

[8] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Eurocrypt*, 2004, pp. 506–522.

[9] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, May 2007, pp. 350–364.

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

[11] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. IEEE ICDCS*, Jun. 2010, pp. 253–262.

[12] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, "Building an encrypted and searchable audit log," in *Proc. NDSS*, 2004, pp. 1–11.

[13] E.-J. Goh *et al.*, "Secure indexes," IACR Cryptol. ePrint Arch., Tech. Rep. 2004/022, 2003, p. 216.

[14] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. TCC*, 2007, pp. 535–554.

[15] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," *J. Comput. Secur.*, vol. 19, no. 3, pp. 367–397, 2011.

[16] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. ACM SIGMOD*, Jun. 2004, pp. 563–574.

[17] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Proc. Adv. Cryptol. CRYPTO*, 2001, pp. 213–229.

[18] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–5.

[19] Z. Shen, J. Shu, and W. Xue, "Preferred keyword search over encrypted data in cloud computing," in *Proc. IEEE/ACM IWQoS*, Jun. 2013, pp. 1–6.

[20] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted data in cloud computing," in *Proc. IEEE ICDCS*, Jun. 2011, pp. 383–392.

[21] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2706–2716, Dec. 2016.

[22] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340–352, Feb. 2016.

[23] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2546–2559, Sep. 2015.

[24] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, "Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Trans. Commun.*, vol. 98, no. 1, pp. 190–200, 2015.

[25] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy*, May 2007, pp. 321–334.

[26] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. ACM CCS*, Nov. 2006, pp. 89–98.

[27] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Proc. Adv. Cryptol.–EUROCRYPT*, 2008, pp. 146–162.

[28] S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "A data outsourcing architecture combining cryptography and access control," in *Proc. ACM Workshop Comput. Secur. Archit.*, Nov. 2007, pp. 63–69.

[29] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[30] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter, "Patient controlled encryption: Ensuring privacy of electronic medical records," in *Proc. ACM Workshop Cloud Comput. Secur.*, Nov. 2009, pp. 103–114.

[31] S. Narayan, M. Gagné, and R. Safavi-Naini, "Privacy preserving EHR system using attribute-based infrastructure," in *Proc. ACM Workshop Cloud Comput. Secur.*, 2010, pp. 47–52.

[32] J. Li, J. Li, X. Chen, C. Jia, and Z. Liu, "Efficient keyword search over encrypted data with fine-grained access control in hybrid cloud," in *Network and System Security*, 2012, pp. 490–502.

[33] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proc. USENIX FAST*, Berlin, Germany: Springer-Verlag, 2003, pp. 29–42.

[34] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," in *Proc. NDSS*, Feb. 2003.

[35] T. Okamoto and K. Takashima, "Hierarchical predicate encryption for inner-products," in *Proc. Adv. Cryptol. ASIACRYPT*, 2009, pp. 214–231.

[36] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

[37] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou, "Privacy-preserving query over encrypted graph-structured data in cloud computing," in *Proc. IEEE ICDCS*, Jun. 2011, pp. 393–402.

[38] D. Boneh, A. Raghunathan, and G. Segev, "Function-private identity-based encryption: Hiding the function in functional encryption," in *Proc. Adv. Cryptol. CRYPTO*, 2013, pp. 461–478.

[39] B. K. Samanthula, W. Jiang, and E. Bertino, "Privacy-preserving complex query evaluation over semantically secure encrypted data," in *Computer Security—ESORICS*, Cham, Switzerland: Springer, 2014.

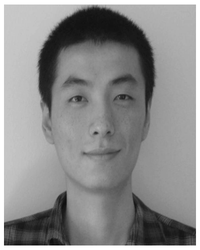
[40] J. Cao, F.-Y. Rao, M. Kuzu, E. Bertino, and M. Kantarcioglu, "Efficient tree pattern queries on encrypted XML documents," in *Proc. Joint EDBT/ICDT Workshops*, Mar. 2013, pp. 111–120.

[41] A. De Caro and V. Iovino, "jPBC: Java pairing based cryptography," in *Proc. 16th IEEE Symp. Comput. Commun.*, Jun. 2011, pp. 850–855.

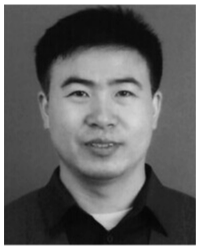
[42] V. Rajkovic *et al. Machine Learning Repository, Nursery Data Set*, accessed on Nov. 2016. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Nursery

[43] R. Kohavi and B. Becker. *Machine Learning Repository, Adult Data Set*, accessed on Nov. 2016. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Adult

[44] *Economic Value Validation Summary—Box Online File Sharing & Collaboration for Enterprise it*, accessed on 2012. [Online]. Available: http://whitepaperbox.com/marketing/economic-value-validation-summary-box-online-file-sharing-collaboration-for-enterprise-it-2/



Zhirong Shen (M'10) received the bachelor's degree from the University of Electronic Science and Technology of China in 2010 and the Ph.D. degree from Tsinghua University in 2016. He is currently an Assistant Professor with Fuzhou University. His current research interests include storage reliability and storage security. He is a member of CCF.



Jiwu Shu (M'09) received the Ph.D. degree in computer science from Nanjing University in 1998. He held a post-doctoral research position at Tsinghua University in 2000. Since then, he has been teaching with Tsinghua University. His current research interests include storage security and reliability, non-volatile memory-based storage systems, and parallel and distributed computing.

Wei Xue (M'10) is currently an Associate Professor with the Center of Earth System Science, Department of Computer Science and Technology, Tsinghua University. His research interests include high performance computing and uncertainty quantification for climate system model. He is a Senior Member of the CCF and member of the ACM.