Keyword Search with Access Control over Encrypted Data in Cloud Computing

Zhirong Shen, Jiwu Shu†, Wei Xue
Department of Computer Science and Technology, Tsinghua University,Beijing 100084, China
Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China
†*Corresponding author:shujw@tsinghua.edu.cn*
zhirong.shen2601@gmail.com,xuewei@tsinghua.edu.cn

***Abstract*—Cloud computing has become an increasingly popular service for data storage and processing. To keep users' data on the cloud from leaking to unauthorized users, probably including the cloud service providers, the data must be stored in an encrypted form. In the meantime, for data intended for sharing, an efficient access control must be provided. A common operation on the data is keyword search. Currently, search operation over encrypted search is performed at the cloud servers and access control for the in-cloud data is usually enforced by users. Separation of the two types of operations can lead to reduced efficiency and compromised privacy for users with a given set of access privileges to search over encrypted cloud data.**

In this paper, we study the problem of keyword search with access control 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. KSAC utilizes a recent cryptographic primitive called HPE to enforce fine-grained access control, perform multi-field query search, and support the derivation of the search capability. Intensive evaluations on real-world dataset are conducted to validate the applicability of the proposed scheme.

I. Introduction

The cloud has become an important platform for data storage and processing. It centralizes essentially unlimited resources and delivers elastic services to end users without performing their own system management and upfront equipment acquisitions. However, data confidentiality protection (to hide the plaintext against the cloud server and other unauthorized users) and data access control (to grant user's access privilege) are usually required so that data owners can confidently store their data onto the cloud.

Encryption is a commonly used method to preserve data confidentiality by storing ciphertext in the cloud. However, it may make traditional approaches designed for plaintext keyword search inapplicable. Aiming at enabling secure and efficient search over encrypted data, *Searchable Encryption* (SE) [1]–[8] receives increasingly more attentions in recent years, in which a query is encrypted as a search capability and a cloud server will return files matching the capability without having to know the keywords both in the capability or in file's encrypted index. However, most of existing SE schemes assume that 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-control policy when searching over encrypted data.

978-1-4799-4852-9/14/$31.00 © 2014 IEEE

There have been a number of works on *access control over encrypted data*. These works can be categorized into two groups, *key-based access control* (KBAC) and *attribute-based access control* (ABAC). KBAC [13], [14] usually assigns each file's decryption key directly to authorized users. When a user receives increasing number of such keys accumulated, its load on the management of the keys can be too high. To reduce the load, ABAC [9], [10] attaches a set of attribute values to a user (or a file) and designs access policy for a file (or a user, respectively). 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 to represent 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 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. An example, in this practice a cloud server may perform search and transfers 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 some of the transferred files have to discarded, which leads to wasted network bandwidth and reduced service efficiency. Actually, there are still a number of critical issues to be addressed for the technique of encrypted search with access control to be widely adopted.

First, for the sake of efficiency and convenience, multi-field search query and fine-grained access control must be supported. Moreover, to harness cloud servers' computation power, we should place the duty of enforcement of access control on servers in the data centers while they are responsible for searching over encrypted data, so that a file sent to a user not only matches the user's search query but also conform to the user's access rights.

Second, most of proposed SE schemes require data owner

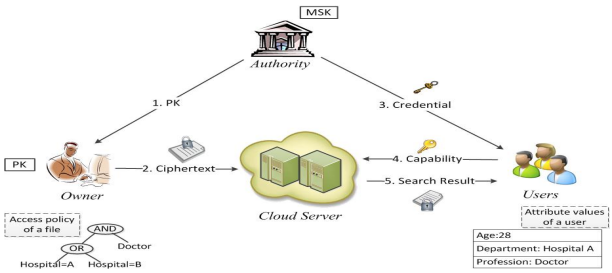


Fig. 1: The Fundamental Framework of KSAC

to handle the search capability derivation for user's interested keywords *every time before search*. This requirement places heavy burden on data owners and can significantly compromise the system scalability. This weakness should be mitigated by allowing user to locally derive the search capability.

In this paper, we systematically study the issue of *keyword search with access control* (KSAC) over encrypted cloud data. The contributions are summarized as follows.

First, we propose a scalable framework as shown in Fig. 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. Each 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 in an interleaving manner. In this way, a user receives the exact data files that match his search query and he has the rights to access.

Second, to enable such a framework, we make a novel use of *Hierarchical Predicate Encryption* (HPE) [19] (a short description is shown in section III and Appendix), to realize the derivation of search capability from credential and a search query. Based on HPE, we propose our scheme named as KSAC, which enables the service of both the query search and access control over multiple fields.

Finally, extensive evaluations have been conducted showing the applicability of KSAC in real scenario.

II. Problem Formulation

A. System Model and Threat Model

In this paper, we consider a cloud-data-sharing system (as shown in Fig. 1) consisting of four entities, i.e., data owners, the authority, data users and the cloud server. *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. The *cloud server* stores the encrypted data and performs search when receiving search capabilities from users. The *authority* is responsible to authenticate user's identity. It issues a set of keys as a credential to represent user's attribute values. *Data users* refers to those who wish to fetch the files according to their interests and access privileges.

Like many previous SE schemes [2]–[4], the cloud server is assumed to be "honest but curious", meaning the server will honestly execute the pre-defined protocol, but it is also "curious" to learn the information about file keywords, user's query and attribute values. The authority is honest to generate the credential according to user's attribute values.

B. Design Goals

The scheme to realize keyword search with access control over encrypted cloud data should promise the following performance and security goals.

Data Confidentiality and Index Privacy: The data confidentiality should be protected against the cloud server and unauthorized users, so that the file content would be seen by authorized users only. Index privacy indicates the cloud server should be unaware of the 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, for example, the access policy and the search query can be represented in *Conjunctive Normal Form* (CNF) over multiple fields, including a class of subset, range and equality relationship on every field.

Efficiency: The system should promise the efficiency for general operations, such as search and credential generation, in practical environment.

C. Notation and Definition

Attribute and Attribute Value. The "*attribute*" is used to characterize the identity of a user. A user's attribute values is denoted as $\mathcal{A} := (A_1 = a_1, ..., A_{n_\mathbb{A}} = a_{n_\mathbb{A}}) \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_{n_\mathbb{A}}$, where a_i and $\mathbb{A}_i$ are an attribute value and the attribute value universe over the *i*-th attribute field A_i respectively, and $n_\mathbb{A}$ denotes the number of attribute fields. For example, a user's attribute values are "Age=37, Profession=Professor", then "37" (a_i) is an attribute value of attribute "Age" (A_i) and the space ($\mathbb{A}_i$) of the attribute "Age" is $[0, 150]$.

Predicate. In our system, we take the predicates class to be $\mathcal{F} = \{f_{\vec{v}} | \vec{v} \in \mathbb{Z}_N^n\}$, where $\vec{v}$ is the predicate vector, for a value vector[1] $\vec{x}$, if $\langle \vec{x}, \vec{v} \rangle = 0$, then we say "$\vec{x}$ is accepted by $\vec{v}$ " and denote it as $f_{\vec{v}}(\vec{x}) = 1$.

Transformation from Logical Form to Vector Form. According to [11], one can represent the logical formula in the vector form. A general case of a CNF formula over multiple fields can be represented as $\mathcal{Q} := (Q_1 \in (a_{1,1}, ..., a_{1,m_1})) \wedge (Q_2 \in (a_{2,1}, ..., a_{2,m_2})) \wedge ... \wedge (Q_e \in (a_{e,1}, ..., a_{e,m_e}))$. It can be converted into the polynomial form with e univariate polynomials as: $f(Q_1, ..., Q_e) := r_1((Q_1 - a_{1,1})...(Q_1 - a_{1,m_1})) + ... + r_e((Q_e - a_{e,1})...(Q_e - a_{e,m_e}))$. The predicate vector can be derived as $\vec{p} := (p_{1,m_1}, ..., p_{1,1}, ..., p_{e,m_e}, ..., p_{e,1}, p_0)$, where $p_0 := \sum_{i=1}^{e}((-1)^{m_i} \cdot r_i \cdot \prod_{j=1}^{m_i} a_{i,j})$ and $p_{i,j}$ denotes the coefficient of Q_i^j in $f(Q_1, ..., Q_e)$. For a value set $\mathcal{V} := (Q_1 = v_1, ..., Q_e = v_e)$, it can be converted into the value vector $\vec{v} := (v_1^{m_1}, ..., v_1, ..., v_e^{m_e}, ..., v_e, 1)$. So that if $\mathcal{V}$ satisfies $\mathcal{Q}$,

[1]In some previous works [11], it is called *attribute vector*. To distinguish it with the term "attribute" used in our work, we denote it as the value vector in this paper.

TABLE I: Some Frequently Used Notations.

Notations	Descriptions
$\mathcal{S}, \vec{S}$	access policy, access policy vector (APV)
W_i, w_i	i-th keyword field, the keyword value of W_i
A_i,a_i	i-th attribute field, the attribute value of A_i
$\mathbb{W}_i, \mathbb{A}_i$	keyword universe of W_i, attribute universe of A_i
$\mathcal{W}, \vec{W}$	keywords of the file, keyword vector (KV)
n_1/n_2	length of 1-st/2-nd level of hierarchy $\vec{\mu}$
$\mathcal{A}, \vec{A}$	the user's attribute values, the user's attribute vector (AV)
$\mathcal{D}_{\vec{A}}$	the credential originated from $\vec{A}$
$\mathcal{Q}, \vec{Q}$	the user's query, the query vector (QV)
$T_{\vec{A},\vec{Q}}$	the search capability generated from $\mathcal{D}_{\vec{A}}$ and $\vec{Q}$
$\langle \vec{x}, \vec{y} \rangle$	the inner-product of vectors $\vec{x}$ and $\vec{y}$

then the equation $\langle \vec{v}, \vec{p} \rangle = f(v_1, ..., v_e) = 0$ establishes with overwhelming probability.

Access Policy Vectors and Attribute Vectors. An *access policy* is a unique logical expression over attribute values to specify the authorized users, where we mainly consider the access policy in CNF formula in this paper. 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.

Keyword and Search Query. The *keyword* refers to the specific terms that characterize file content. For a file F, its keyword set can be expressed as $\mathcal{W} := (W_1 = w_1, ..., W_{n_\mathbb{W}} = w_{n_\mathbb{W}}) \in \mathbb{W}_1 \times \cdots \times \mathbb{W}_{n_\mathbb{W}}$, where w_i and $\mathbb{W}_i$ are the keyword value and keyword value universe over the i-th field W_i respectively and $n_\mathbb{W}$ is the number of the keyword fields. A "*search query*" represents the user's interest, and is expressed in the logical form of keywords. The search query and keywords can be converted into the predicate vector called *query vector* (QV) and the value vector called *keyword vector* (KV) after the transformation.

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=2012/1/1 $\wedge$ Topic=meeting", then the search capability would include the logical information of "Age=27, Profession=professor, Date=2013/1/1 $\wedge$ Topic=meeting".

III. System description of KSAC

In this section, we will describe the main algorithms of KSAC and show how to make use of hierarchical predicate encryption (HPE) to simultaneously support fine-grained access control and multi-keyword query over encrypted data. We will start with the introduction of HPE first.

1. An Introduction to HPE. Hierarchical predicate encryption (HPE) [19] is an cryptographic primitive that supports delegation of *predicate encryption* (PE) [11]. 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, so that the decryption will succeed if $\langle \vec{p}_l, \vec{v} \rangle = \langle \vec{p}_{l+1}, \vec{v} \rangle = 0$. Appendix gives a more detailed description of HPE.

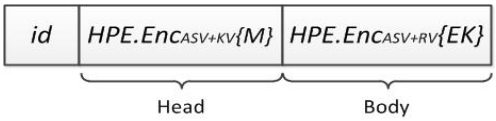


Fig. 2: The Format of An Encrypted Index. The "ASV" and "KV" represent the vectors transformed from access policy and keywords, respectively. "RV" is a selected random vector.

2. The Design of Index Format. Before giving the system description of KSAC, we first design the format of the encrypted index as shown in Fig. 2. The index information includes the specified access policy, the representative 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, ensuring only the users whose attribute values satisfy the access policy can recover *EK*. He further produces the "*head*" component using both the representative keywords and the access policy, to guarantee the file can be retrieved only when the keywords match with the query and the user's attribute values satisfy the access policy.

3. System Description. Our proposed keyword search with access control scheme is shown in Algorithm 1, which contains 6 main algorithms.

• KSAC.Setup. The authority chooses a security parameter λ and a format of hierarchy $\vec{u}$, invokes HPE.Setup, and outputs a public key PK and a master secret key MSK. PK will be 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 ($i := 1, 2$). The first vector space is the attribute space, while the second space is the keyword space.

• KSAC.BuildIndex. The owner converts the access policy and keywords into the *access policy vector* (ASV) and *keyword vector* (KV), chooses a random vector (RV), and builds the encrypted index by calling HPE.Enc.

• KSAC.GenCre. According to user's attribute values $\mathcal{A}$, the authority converts $\mathcal{A}$ into the *attribute vector* (AV) first and then issues the corresponding *credential* $D_{\vec{A}} := (\mathbf{k}^*_{1,0}, \mathbf{k}^*_{1,1}, \mathbf{k}^*_{1,2}, \mathbf{k}^*_{1,n_1+1}, ..., \mathbf{k}^*_{1,n_1+n_2})$ by calling HPE.GenKey.

• KSAC.DeriCap. For an interested 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 realization of the algorithm HPE.Delegate$_1$, but just compute $\mathbf{k}^*_{2,0}$ to make it serve as the capability.

• KSAC.Search After receiving the capability $T_{\vec{A},\vec{Q}}$, the cloud server calls HPE.Dec. If the result is in the form of $I_{\mathbb{G}_T}$, where $I_{\mathbb{G}_T}$ is the identity of the group $\mathbb{G}_T$ used in HPE, then it indicates the condition $\langle \vec{Q}, \vec{W} \rangle = \langle \vec{A}, \vec{S} \rangle = 0$ establishes, meaning the file's keywords match with the search query (i.e.,

Algorithm 1: The Primary Design of KSAC.

- KSAC.Setup($1^\lambda, \vec{u}$): Invoke 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 designed 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 $C^{head} := (\mathbf{C}_1^{head}, C_2^{head})$:=HPE.Enc$(PK, (\vec{S}, \vec{W}), I_{\mathbb{G}_T})$ and $C^{body} := (\mathbf{C}_1^{body}, C_2^{body})$:=HPE.Enc$(PK, (\vec{S}, \vec{R}), EK)$
- KSAC.GenCre($PK, MSK, \mathcal{A}$): Validate a user's identity, convert $\mathcal{A}$ into the attribute vector $\vec{A} := (a_1, ..., a_{n_1})$, and generate the credential $D_{\vec{A}} := (\mathbf{k}^*_{1,0}, \mathbf{k}^*_{1,1}, \mathbf{k}^*_{1,2}, \mathbf{k}^*_{1,n_1+1}, ..., \mathbf{k}^*_{1,n_1+n_2})$:=HPE.GenKey$(PK, MSK, \vec{A})$
- KSAC.DeriCap($\mathcal{D}_{\vec{A}}, \mathcal{Q}, PK$): Transform the query $\mathcal{Q}$ into the query vector $\vec{Q} := (q_1, ..., q_{n_2})$, obtain $\mathbf{k}^*_{2,0}$ by calling HPE.Delegate$_1(PK, D_{\vec{A}}, \vec{Q})$, and return $T_{\vec{A},\vec{Q}} := \mathbf{k}^*_{2,0}$.
- KSAC.Search($T_{\vec{A},\vec{Q}}, C^{head}, PK$): Compute 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 HPE.Dec($C^{body}, \mathcal{D}_{\vec{A}}, PK$).

$\langle \vec{Q}, \vec{W} \rangle = 0$) 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 match with 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. 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. This indicates that *EK* can only be unlocked by the authorized user's credential. An unauthorized user's attempt to recovery *EK* will be rejected by $\vec{S}$ and the cloud server's attempt to unlock *EK* by using an authorized user's search capability will be refused by $\vec{R}$.

4. Analysis of KSAC.

1) Data Confidentiality and Index Privacy: In KSAC, the plaintext is first encrypted by the symmetric algorithm and the symmetric key EK is then defended by HPE. Thus, the data confidentiality is ensured by the symmetric algorithm and HPE together.

Meanwhile, HPE is selective attribute-hiding against chosen-plaintext attacks, meaning 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 two 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, 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 [19], 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 $\mathcal{S}$ and keywords $\mathcal{W}$ unless it obtains $T_{\vec{A},\vec{Q}}$ where $(\vec{A}, \vec{S}) = 0$, $(\vec{Q}, \vec{W}) = 0$, thus KSAC can achieve the selective attribute-hiding security as HPE to guarantee index privacy.

2) Fine-grained Access Control and Multi-Field Search Query: In KSAC, the owner is able to define and enforce the access control depending on ASV, which is converted from the logical combination of the authorized attribute values over each attribute field. Meanwhile, KSAC allows users to employ a query vector to express the search query represented in CNF among multiple dimensions, including subset, range and equality query on every dimension.

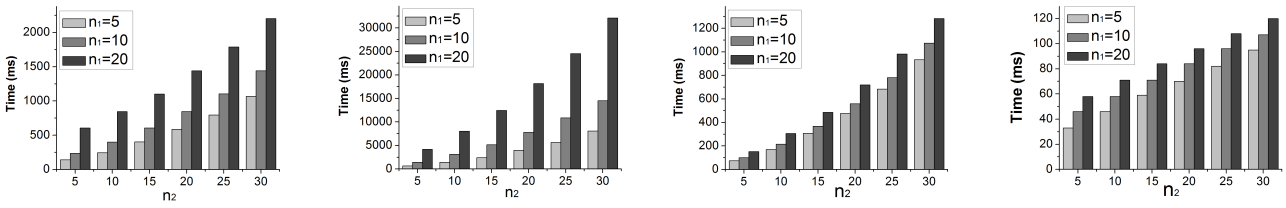
IV. Performance Evaluation

We fully realize KSAC by using java Pairing-Based Cryptography (jPBC) Library [12]. The server which is intended to carry out the experiment is equipped with 2.10GHZ Intel® Core 2 Duo CPU and 4GB RAM. The operating system run on the server is Ubuntu (version: 11.04) and the kernel is Linux Ubuntu 2.6.38-8-generic.

We compare KSAC with *Predicate Encryption* (PE) [11] on the metric of access control under the same length of APV, since PE can support CNF formula in vector form and provide selective attribute-hiding security, which are the same with KSAC. The three balance primes in PE are selected with 342 bits each to equivalently achieve the same security strength provided by the group whose order is 160-bits. Moreover, we also compare KSAC with MRQED [2] (we assume $d = 1$) on the efficiency of keyword search under the same length of KV.

We define a format of hierarchy $\vec{u}$ with 2 levels; the first level is access control vector space (i.e., APV/AV), and the second level is query vector space (i.e., QV/KV). We use n_1 to denote the length of the first space and use n_2 to represent the length of the second level. Meanwhile, we also define $n_0 = n_1 + n_2 + 3$, where n_0 is the dimensions of the vector space defined in KSAC over Elliptic Curve Cryptography (ECC).

We select attributes "Sex" and "Occupation" with 2 values and 14 values respectively from the "Adult Data Set" [17] in UCI Machine Learning Repository [15] to characterize user's attribute values. According to the transformation in Section II, $n_1 := \sum_{i=1}^{2} n_{\mathbb{A}_i} + 1 := 17$. We then use the "Car Evaluation Data Set" [16] to denote the data files, where



(a) BuildIndex (b) GenCre (c) DeriCap (d) Search

Fig. 3: The Performance of KSAC

attributes and corresponding values are treated as keyword fields and keywords respectively. There are six attributes in the dataset, where each of the first three attributes has four values while each of the other three attributes have three values. Thus the length of QV/KV $n_2 := \sum_{i=1}^{6} n_{\mathbb{W}_i} + 1 = 22$, and $n_0 := n_1 + n_2 + 3 := 42$.

1. Experiment Setup. We adopt type A elliptic parameter [12], which is constructed on the curve $y^2 = x^3 + x$ over the field $\mathbb{F}_q$, 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 65B in compressed form in KSAC and MRQED if the size of the base field for the group is 512 bits when q is 20B, thus the size of PK and MSK in KSAC are $65[n_0(n_0 - 1) + 3]$B and (65+20)n_0^2=85n_0^2B respectively. If n_1=17,n_2=22, then it needs 109.5KB and 146.4KB to keep PK and MSK respectively.

For PE, an element in $\mathbb{G}$ can be represented by 130B and it needs 43B to represent the order of the subgroups, thus the size of *PK* is 130(2n_1+4)B, and the size of MSK is [43×3+130(2+2n_1)]B. If n_1=17, then it needs 4.8KB and 4.7KB to keep PK and MSK in PE, respectively. Though it requires a little more storage space to maintain the PK and MSK of KSAC, it is less an issue today because of the lower price of the storage space. Thus it needs $65[8(n_2 - 1) + 1]$B to keep PK and MSK in MRQED and the size of each of them is 10.7KB when n_2=22.

2. Build Index. For index construction, we mainly consider the time to create C^{head} and C^{body} and show the averaged result in Fig 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}, i.e., $(65n_0 + 1)$B. So when n_1=17, n_2=22, the creation time and the size of head/body part are 1.34s and 2.7KB respectively.

In addition, the size of the ciphertext in PE is $130(2n_1 + 2)$B, thus when n_1=17, the encryption time and the size of head/body component are 0.42s and 4.6KB respectively. For MRQED, it takes $O(n_2)$ exponentiations and $65[4(n_2 - 1) + 2]$B to calculate and keep the ciphertext. When n_2=22, the encryption time and the size of C^{head}/C^{body} are 1.0s and 5.5KB, respectively.

According to the above comparison, to build an encrypted index, KSAC needs a bit more time than both of PE and MRQED. However, building an index is still a one-time operation and is not very significant to evaluate the total performance of a scheme. In the environment with keyword search and access control, the most frequently performed operations are the search capability derivations and the search operations, which should receive special attention.

3. Credential Generation. For credential generation, we test many cases when n_1 and n_2 change and show the averaged results in Fig. 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, such as when the user applies to be registered, 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 [2] can decrease the time of exponential operation from 6.4ms to 30us. Furthermore, the credential size is $65(n_0 + 2n_0 + n_2 \cdot n_0)$, so when n_1=17, n_2=22, the credential size and the generation time are 66.7KB and 16.5s, respectively.

4. Search Capability Derivation. During the generation of $T_{\vec{A},\vec{Q}}$, we need not implement the HPE.Delegation algorithm completely, but only produce $\mathbf{k}^*_{2,0}$ (i.e., $T_{\vec{A},\vec{Q}}$) to make it serve as $T_{\vec{A},\vec{Q}}$. The evaluated results are shown in Fig 3(c). We can observe that when n_1=17, n_2=22, deriving a search capability calls for about 0.75s, which is quite applicable in the real scenario. For the storage overhead, a search capability takes up $65n_0$B, which also can be regarded as the communication overhead. So when n_1=17, n_2=22, it needs about 2.7KB to reposit a valid search capability, which will not bring much storage burden to the user. The user can keep the frequently used search capabilities to reduce the time cost in multiple search capability generations.

When applying PE, if n_1=17, the generation of a secret key requires about 3.84s, which is 5.12 times of that of KSAC. In addition, PE system requires the owner to be responsible for the secret key generation. This setting will easily make the owner become the bottleneck of the system, once the access requests increase sharply. Against that, the capability generation in KSAC is removed to users to migrate the computation burden, which will benefit the scalability of the whole system. For MQRED, when $n_2 = 22$, the time and the capability size are 0.99s and 6.7KB, which are about 1.32 times and 2.48 times of those of KSAC.

From the evaluations above, we can draw the conclusion that data user in KSAC has to afford most of the burden during the whole procedure of capability derivation, which makes KSAC more scalable when compared with both PE and MRQED.

5. Search. We present the average search time per file under different cases in Fig. 4(d). The computation complexity is

linear with n_0 since HPE.Dec calls for n_0 pairing operations and $(n_0 - 1)$ multiplication operations. When n_1=17, n_2=22, it takes about 0.11s to judge whether a file satisfies the search capability.

For PE, if n_1=17, the time to check user's access privilege after preprocessing is about 1.3s, near 11.8 times of ours. For MQRED, it takes $O(n_2)$ pairing operations. When n_2=22, the search time is 0.32s, about 2.9 times of ours. Since the search operation is one of the operations that are most frequently performed, the advantage of KSAC on search operation also make it outperform than its competitors in real scenario.

V. Related work

Searchable Encryption. Song et al. [3] proposed the first symmetric-key based searchable encryption scheme. Goh et al. [6] presented secure indexed over encrypted data by employing Bloom Filter. Wang et al. [4] introduced secure ranked keyword search based on "order-preserving encryption". In the public key setting, Boneh et al. [1] first introduced the searchable encryption scheme by using bilinear mapping. Water et al. [5] fulfilled searchable audit log using symmetric encryption and IBE respectively. Golle et al. [8] developed two schemes to realize conjunctive keyword search over encrypted data. Shi et al. [2] realized multi-dimensional range query over encrypted data. Shen et al. [24] investigated the encrypted search with preference by utilizing Lagrange polynomial and secure inner-product computation. Li et al. [18] solved the problem of authorized private keyword search, which only achieved LTA-level authorization which was far coarser than user-level access control. [21] partially refers to the similar problem in KSAC, but it would easily cause considerable complexity of keys management.

Access Control over Encrypted Data. Bethencourt et al. [9] proposed CP-ABE to regulate the user's privilege to the shared data. As the dual problem of CP-ABE [9], KP-ABE [10] embeded the access policy in the user's keys while the data were labeled with attributes. HVE [7] and PE [11] were the new tools which could be used to perform access control over encrypted data, and they all employed composite-order groups. Sabrina et al. [20] utilized over encryption to realize access control. Benaloh et al. [21] considered the security problem of EHR, but it only supported single keyword search, and did not achieve flexible access control. Narayan et al. [22] just combined bABE and PEKS [23] together to realize a patient-centric EHR management system, indicating even the users with different attribute values would receive the same files for the same query. Meanwhile, it still forced owners to process user's every search request.

VI. Acknowledgments

We would like to thank Elaine Shi for her helpful discussion. This work is supported by the National Natural Science Foundation of China (Grant No. 61232003, 61327902), the National High Technology Research and Development Program of China (Grant No. 2012AA011003), Shanghai Key Laboratory of Scalable Computing and Systems, Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and Tsinghua University Initiative Scientific Research Program.

VII. Conclusions

In this paper, we address keyword search with access control in encrypted cloud data, where the data are tied with access policies and keywords, and allow the searches by multiple users whose access privileges are specified. We propose a scalable framework that allows user to locally derive the search capability by utilizing both his credentials and a search query. We then utilize HPE to realize this framework and present our design named KSAC. KSAC realizes the fine-grained access control and multi-field keyword search. Finally, the intensive evaluations demonstrate the applicability of KSAC.

References

[1] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. *Public Key Encryption with keyword search.* In Proc. of EUROCRYPT, 2004.

[2] E. Shi, J. Bethencourt, T. Chan, D. Song, and A. Perrig. *Multi-Dimensional Range Query over Encrypted Data.* In Proc. of IEEE S&P, 2007.

[3] D.Song, D.Wagner, and A.Perrig. *Practival Techniques for Searches on Encrypted Data*. In Proc. of IEEE S&P, 2000.

[4] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. *Secure Ranked Keyword Search over Encrypted Cloud Data*. In Proc. of ICDCS'10, 2010.

[5] B. Waters,D. Balfanz, G. Durfee, D.K. Smetters. *Building an Encrypted and Searchable Audit Log*. In Proc. of NDSS 2004.

[6] E.Goh. *Secure Indexes*. Cryptology ePrint Archive, 2003, http://eprint.iacr.org/2003/216.

[7] D. Boneh and B. Waters. *Conjunctive, Subset, and Range Queries on Encrypted Data*. in Proc. of TCC, 2007, pp. 535-554.

[8] P. Golle, J. Staddon, and B. Waters. *Secure Conjunctive Keyword Search Over Encrypted Data*. in Proc. of ACNS, 2004.

[9] J. Bethencourt,A. Sahai, B. Waters. *Ciphertext-Policy Attribute-Based Encryption*, In Proc. of IEEE S&P, 2007.

[10] V. Goyal, O. Pandey, A. Sahai, and B. Waters. *Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data*. In Proc. of ACM CCS, 2006.

[11] J. Katz, A. Sahai, and B. Waters. *Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products*. in Proc. of EUROCRYPT, 2008.

[12] Angelo De Caro. The Java Pairing Based Cryptography Library(jPBC) http://gas.dia.unisa.it/projects/jpbc/index.html

[13] E. Goh, H.Shacham, N. Modadugu, and D. Boneh. *Sirius: Securing Remote Untrusted Storage*. In Proc. of NDSS,2003

[14] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, K. Fu. *Scalable Secure File Sharing on Untrusted Storage*. In Proc. of FAST, 2003.

[15] A. Frank and A. Asunicion. *UCI machine learing repository*, 2010.

[16] M. Bohanec. Machine Learning Repository, Car Evaluation Data Set. http://archive.ics.uci.edu/ml/datasets/Car+Evaluation.

[17] R. Kohavi and B. Becker, http://archive.ics.uci.edu/ml/datasets/Adult.

[18] M. Li, S. Yu, N. Cao and W. Lou. *Authorized Private Keyword Search over Encrypted Personal Health Records in Cloud Computing*. In Proc. of ICDCS,2011.

[19] T. Okamoto and W.Takashima. *Hierarchical Predicate Encryption for Inner-Products* In Proc. of ASIACRYPT, 2009.

[20] S. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati. *A Data Outsourcing Architecture Combining Cryptography and Access control.* In Proc. of CSAW, 2007

[21] J. Benaloh, M.Chase, E. Horvitz, and K. Lauter. *Patient Controlled Encryption: Ensuring Privacy of Electronic Medical Records.* In ACM CCSW, 2009

[22] S. Narayan, M. Gagne, and R. Safavi-Naini. *Privacy preserving EHR system using attribute-based infrastructure*. In ACM CCSW, 2010.

[23] L. Fang, W. Susilo, C. Ge, and J. Wang. *A Secure Channel Free Public Key Encryption with Keyword Search Scheme without Random Oracle.* In Proc.of CANS, 2009.

[24] Z. Shen, J. Shu, and W. Xue. *Preferred Keyword Search over Encrypted Data in Cloud Computing.* In Proc.of IWQoS, 2013.