Verifiable Attribute-based Keyword Search over Outsourced Encrypted Data

Qingji Zheng† Shouhuai Xu† Giuseppe Ateniese ‡
† University of Texas at San Antonio, USA
‡ Sapienza University of Rome, Italy and Johns Hopkins University, USA

***Abstract*—It is quite common nowadays for data owners to outsource their data to the cloud. However, since the cloud is not fully trusted, the outsourced data should be encrypted, which brings a range of problems, such as: How can authorized data users search over a data owner's outsourced encrypted data? How should a data owner grant search capabilities to data users? How can data users be assured that the cloud faithfully executed the search operations? Towards ultimately addressing these problems, in this paper we propose a novel cryptographic scheme, called *verifiable attribute-based keyword search* (VABKS). This scheme allows a data user, whose attributes or credentials satisfy a data owner's access control policy, to (i) search over the data owner's outsourced encrypted data, (ii) outsource the tedious search operations to the cloud, and (iii) verify whether the cloud has faithfully executed the search operations. We define VABKS's security properties, and present concrete constructions that are proven to possess these properties. Performance evaluation shows that the proposed schemes are practical.**

I. Introduction

Cloud computing allows data owners to use massive data storage and vast computation capability at low price. Despite the benefits, data outsourcing deprives data owners of direct control over their outsourced data. To alleviate such concerns, it is natural for data owners to encrypt their outsourced data. However, encrypting outsourced data brings a range of problems, especially when the outsourced data needs to be shared with others. In this paper, we address some of the problems that are encountered when the data owner needs to enforce access control over its outsourced encrypted data, while the data user needs to outsource the search operations to the cloud for performance reasons. To the best of our knowledge, existing solutions are not sufficient for the goals we aim to achieve.

A. Our Contributions

We propose a novel cryptographic schem, which allows a data user with proper attributes (according to a data owner's access control policy) to (i) search over the data owner's outsourced encrypted data, (ii) outsource the actual search operations to the cloud, and (iii) verify whether the cloud has faithfully executed the search operations. This is achieved through a novel cryptographic scheme, called *verifiable attribute-based keyword search* (VABKS). This scheme allows data owners to control the search and use of its outsourced encrypted data according to its access control policy, while the legitimate data users can outsource the often costly search operations to the cloud and can verify whether the cloud has faithfully executed the search operations.

In addition to formally define the security properties of VABKS, we propose the design of VABKS schemes in a modular fashion, by using attribute-based encryption, bloom filter, digital signature, and a newly introduced building-block called attribute-based keyword search (ABKS), which may be of independent value. The novel application of bloom filter and digital signature allows data users to *efficiently* verify that the cloud honestly conducted search operations over outsourced encrypted data. Experimental evaluation shows that the VABKS (and ABKS) schemes are practical.

B. Related Work

To the best of our knowledge, there is no solution is sufficient for what we want to achieve. Nevertheless, the relevant techniques are briefly reviewed as follows.

Keyword Search over Encrypted Data. Existing solutions for keyword-based search over encrypted data can be classified into two categories: searchable encryption in the symmetric-key setting (e.g., [1]–[9]) and searchable encryption in the public-key setting (e.g., [10]–[14]). In these solutions, the data and the associated keywords are encrypted with symmetric keys or public keys before being outsourced to the server/cloud. The data owner generates some tokens that can be used by data users to search over the encrypted data. Several variants(e.g., [15]–[18]) have been proposed to support complex queries. The work [4], [19] considered searchable encryption in the multi-users setting, where the data owner can enforce access control policies by distributing (stateful) secret keys to authorized users directly and revoke them if necessary. However, these solutions do not solve the problem VABKS aims to solve because (1) part of solutions require interactions between data users and data owners (or trusted authorities, e.g., trapdoor generation entity [14]) to grant search capabilities, and (2) they assume that the server can faithfully execute search operations. Compared with these solutions, VABKS allows data users with proper attributes to issue search tokens so that the cloud can perform keyword search on their behalf, *without* interacting with data owners, and can assure that the cloud honestly executes search operations even if it is untrusted. Note that while ABKS can be cast into the framework of predicate encryption [20], [21], VABKS provides the novel feature of verifiability.

Attribute-Based Encryption (ABE). ABE allows entities with proper credentials to decrypt a ciphertext in question [22]. It has two variants depending on how access control is enforced. Specifically, key-policy ABE (KP-ABE), where the decryption key is associated with an access control policy [23], and ciphertext-policy ABE (CP-ABE), where the ciphertext is associated with an access control policy [24]. ABE has become the de facto standard for enforcing access control policies via cryptographic means, and has been enriched with various features (e.g., [25]–[28]). In this paper, we propose extending ABE with the novel feature of keyword search, namely *attribute-based keyword search*, by which keywords are encrypted together with some access control policies so that only data users with the right credentials can generate proper tokens for searching on encrypted data. This effectively prevents data owners from knowing keywords a data consumer is searching for and requires no interaction between data users and data owners/trusted authorities. Note that although [14] allows data users to obtain search token from data owners/trusted authorities without revealing keywords, data owners/trusted authorities must always be online and available.

Verifiable Keyword Search. Recently, verifiable keyword search solutions have been proposed in [29]–[31], where each keyword is represented as a root of a polynomial. It is possible to check whether keywords are in the data set by evaluating the polynomial on them and verifying that its output is zero. However, these approaches work only if keywords are sent in the clear to the cloud, thus they are not suitable for the cloud data sharing scenario. Verifiable keyword search in the symmetric-key setting was investigated in [32], which is *not* secure in the public-key setting because the attacker can easily infer keywords directly from the small alphabet space via keyword guessing attack.

Paper Organization: Section II reviews some cryptographic preliminaries. Section III defines ABKS and its security properties, presents the KP-ABKS and CP-ABKS schemes and analyzes their security. Section IV defines VABKS and its security properties, presents a VABKS scheme and analyzes its security. Section V evaluates the performance of the ABKS and VABKS schemes. Section VI concludes the paper.

II. Preliminaries

Given a set S, let $a \leftarrow S$ denote selecting an element a from S uniformly at random. Let $\mathsf{U} = \{\mathsf{at}_1, \ldots, \mathsf{at}_n\}$ be the set of all relevant attributes and $||$ denote the concatenation.

A. Cryptographic Assumption

Let p be an ℓ-bit prime and G, G_T be cyclic groups of prime order p with generators g, g_T respectively. Let e be a bilinear map: $e : G \times G \rightarrow G_T$ satisfying: (i) $\forall a, b \leftarrow \mathbb{Z}_p$, $e(g^a, g^b) = e(g, g)^{ab}$, (ii) $e(g, g) \neq 1$, and (iii) e can be computed efficiently.

Decisional Linear Assumption (DL). Given $(g, f, h, f^{r_1}, g^{r_2}, Q)$ where $g, f, h, Q \leftarrow G$, $r_1, r_2 \leftarrow \mathbb{Z}_p$, this assumption says that any probabilistic polynomial-time algorithm $\mathcal{A}$ determines $Q \stackrel{?}{=} h^{r_1+r_2}$ with a negligible advantage in security parameter ℓ, where the advantage is defined as

$$|\mathsf{Pr}[\mathcal{A}(g, f, h, f^{r_1}, g^{r_2}, h^{r_1+r_2}) = 1] - \mathsf{Pr}[\mathcal{A}(g, f, h, f^{r_1}, g^{r_2}, Q) = 1]|.$$

Generic Bilinear Group [33]. The generic bilinear map is described as follows: Let ψ_0, ψ_1 be two random encodings of the additive group $\mathbb{Z}_p^+$, such that ψ_0, ψ_1 are injective maps from $\mathbb{Z}_p^+$ to $\{0,1\}^m$, where $m > 3\log(p)$. Let $G = \{\psi_0(x) | x \in \mathbb{Z}_p\}$ and $G_T = \{\psi_1(x) | x \in \mathbb{Z}_p\}$. In addition, there is another oracle to compute $e : G \times G \rightarrow G_T$. G is referred to as a generic bilinear group. Let g denote $\psi_0(1)$, g^x denote $\psi_0(x)$, $e(g, g)$ denote $\psi_1(1)$, and $e(g, g)^y$ denote $\psi_1(y)$.

Pseudorandom Generator [34]. A pseudorandom generator $H : \{0,1\}^\ell \rightarrow \{0,1\}^m, \ell < m$ is a deterministic algorithm that takes as input a short random seed and generates a long string that is pseudorandom.

B. Bloom Filter for Membership Query

A Bloom filter [35] is a data structure suitable for representing a set of items while allowing membership queries. Initially, a m-bit Bloom filter is an array of m bits, which are set to 0, and uses k independent hash functions (i.e. universal hash) $H'_1, \ldots, H'_k$ of range $0, \ldots, m-1$. For each element $w \in S = \{w_1, \ldots, w_n\}$, the bits $H'_j(w)$ are set to 1 for $1 \leq j \leq k$. The locations in the m-bit Bloom filter can be set to 1 more than once, but only the first change has an effect. To check whether w is an element of S, it is enough to verify that all $H'_j(w)$, for $1 \leq j \leq k$, are set to 1. If not, w is certainly not an element of S; otherwise, w might be a member of S with high probability (a false-positive rate should be considered). In the m-bit Bloom filter, the false positive rate can be calculated given the assumption that hash functions are perfectly random: After n elements are hashed into the m-bit Bloom filter, the false positive rate is $(1 - (1 - \frac{1}{m})^{km})^k \approx (1 - e^{-kn/m})^k$. We can calculate the optimal number of hash functions minimizing the false positive rate, which is $k = (\ln 2)m/n$. Thus, the false positive rate in this case is $(0.6185)^{m/n}$. A m-bit Bloom filter has two associated algorithms:

- $\mathsf{BF} \leftarrow \mathsf{BFGen}(\{H'_1, \ldots, H'_k\}, \{w_1, \ldots, w_n\})$: This algorithm generates a m-bit Bloom filter by hashing the data set $\{w_1, \ldots, w_n\}$ with $\{H'_1, \ldots, H'_k\}$.
- $\{0,1\} \leftarrow \mathsf{BFVerify}(\{H'_1, \ldots, H'_k\}, \mathsf{BF}, w)$: This algorithm returns 0 if w is not an element of the data set, and 1 otherwise.

C. Access Trees for Representing Access Control Policies

Access trees are convenient for representing access control policies [23]. In an access tree, a leaf is associated with an attribute and an inner node represents a threshold gate (of which AND and OR gates are special cases). Let num_v be the number of children of the node v, and label the ordered children as 1 to num_v. Let k_v be the threshold value associated with an inner node v where $1 \leq k_v \leq \mathsf{num}_v$. Let $\mathsf{parent}(v)$ denote the parent of the node v, $\mathsf{ind}(v)$ denote the label associated with node v, $\mathsf{att}(v)$ denote the attribute associated with leaf node v, $\mathsf{lvs}(\mathsf{T})$ denote the set of leaves of the access

tree T, and T_v denote the subtree of T rooted at the node v (thus, $\mathsf{T}_{\mathsf{root}} = \mathsf{T}$).

Given an attribute set $\mathsf{Atts} \subseteq \mathsf{U}$, we use a function F such that $F(\mathsf{Atts}, \mathsf{T}_v) = 1$ means Atts satisfies the access control policy represented by the subtree T_v. The evaluation of $F(\mathsf{Atts}, \mathsf{T}_v)$ can be performed iteratively as follows:

- In the case v is a leaf: If $\mathsf{att}(v) \in \mathsf{Atts}$, set $F(\mathsf{Atts}, \mathsf{T}_v) = 1$; otherwise, set $F(\mathsf{Atts}, \mathsf{T}_v) = 0$.
- In the case v is an inner node with children $v_1, \ldots, v_{\mathsf{num}_v}$: If there exists a subset $I \subseteq \{1, \ldots, \mathsf{num}_v\}$ such that $|I| \geq k_v$ and $\forall\ j \in I, F(\mathsf{Atts}, \mathsf{T}_{v_j}) = 1$, set $F(\mathsf{Atts}, \mathsf{T}_v) = 1$; otherwise, set $F(\mathsf{Atts}, \mathsf{T}_v) = 0$.

Given an access tree T, we denote the algorithm for distributing a secret s according to the access tree T as:

$$\{q_v(0) | v \in \mathsf{lvs}(\mathsf{T})\} \leftarrow \mathsf{Share}(\mathsf{T}, s).$$

The algorithm operates as follows. Generate a polynomial q_v for each node v in T with the respective threshold value k_v in a top-down fashion according to T (for each leaf node, its threshold value is naturally 1).

- If v is the root of T ($v = \mathsf{root}$), set $q_v(0) = s$ and randomly pick $k_v - 1$ coefficients for polynomial q_v.
- If v is a leaf of T, set $q_v(0) = q_{\mathsf{parent}(v)}(\mathsf{ind}(v))$.
- If v is an inner node, set $q_v(0) = q_{\mathsf{parent}(v)}(\mathsf{ind}(v))$ and randomly select $k_v - 1$ coefficients for polynomial q_v.

When the algorithm stops, each leaf v is associated with a value $q_v(0)$, which is the secret share of s.

Given an access tree T and a set of values $\{E_{u_1}, \ldots, E_{u_m}\}$, where $u_1, \ldots, u_m$ are leaves of T such that $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}) = 1$, $E_{u_j} = e(g,h)^{q_{u_j}(0)}$ for $1 \leq j \leq m, g, h \in G$, e is a bilinear map and $q_{u_1}(0), \ldots, q_{u_m}(0)$ are secret shares of s according to T, we denote the algorithm for reconstructing $e(g,h)^s$ as

$$e(g,h)^s \leftarrow \mathsf{Combine}(\mathsf{T}, \{E_{u_1}, \ldots, E_{u_m}\}).$$

The algorithm operates as follows. Execute the following steps with respect to node v in a bottom-top fashion according to T.

- If $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_v) = 0$, then continue.
- If v is a leaf and $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_v) = 1$, then $E_v = E_{u_j}(0) = e(g,h)^{q_{u_j}(0)}$ where $v = u_j$ for some j.
- Otherwise, v is an inner node and $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_v) = 1$. Then for v's children nodes $\{v_1, \cdots, v_{\mathsf{num}_v}\}$, there should exist a set of indices S, such that $|S| = k_v$, such that $j \in S, F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_{v_j}) = 1$. Let $\Delta_{v_j} = \prod_{l \in S, l \neq j} \frac{-j}{l-j}$ so that

$$E_v = \prod_{j \in S} E_{v_j}^{\Delta_{v_j}} = \prod_{j \in S} (e(g,h)^{q_{v_j}(0)})^{\Delta_{v_j}} = e(g,h)^{q_v(0)}$$

When the algorithm stops, the root of T is with a value $E_{\mathsf{root}} = e(g,h)^{q_{\mathsf{root}}(0)} = e(g,h)^s$.

III. Attribute-Based Keyword Search

In order to effectively grant search capability over encrypted keywords, it is expected that data owners can specify access control policies when they encrypt keywords. After that it requires no further interaction between data owners and data users, which is similar to the intuition of attribute-based encryption. This inspire us to device a new cryptographic scheme, called Attribute-Based Keyword Search (ABKS). It allows data owners to bind keyword ciphertexts with access control policies, so that authorized data users, whose attributes (or credentials) satisfy access control policies, can conduct keyword search (i.e. generating proper tokens) over keyword ciphertexts.

To ease the presentation, we use I_{Enc} to denote the input to the encryption function Enc, and I_{KeyGen} to denote the input to the key generation function KeyGen. In the case of ciphertext-policy ABKS (CP-ABKS), I_{Enc} and I_{KeyGen} are an access structure and an attribute set, respectively. In the case of key-policy ABKS (KP-ABKS), I_{Enc} and I_{KeyGen} are an attribute set and an access structure, respectively. Let $F(I_{\mathsf{KeyGen}}, I_{\mathsf{Enc}}) = 1$ denote that I_{KeyGen} satisfies I_{Enc} in CP-ABKS or I_{Enc} satisfies I_{KeyGen} in KP-ABKS.

A. ABKS *Definition and Security*

Definition 1: An ABKS scheme consists of the following algorithms:

- $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Setup}(1^\ell)$: This algorithm initializes the public parameter pm and generates a master key mk.
- $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: Taking as input the master key mk and I_{KeyGen}, this algorithm outputs credential sk.
- $\mathsf{cph} \leftarrow \mathsf{Enc}(w, I_{\mathsf{Enc}})$: Taking as input keyword w and I_{Enc}, this algorithm outputs a ciphertext cph.
- $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$: Taking as input credential sk and keyword w, this algorithm outputs a search token tk.
- $\{0,1\} \leftarrow \mathsf{Search}(\mathsf{cph}, \mathsf{tk})$: This algorithm determines whether cph contains the keyword w specified by tk. It returns 1 if so, and 0 otherwise.

The correctness of the ABKS scheme requires that, given $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Setup}(1^\ell)$, $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$ and $F(I_{\mathsf{KeyGen}}, I_{\mathsf{Enc}}) = 1$, for any w, $\mathsf{cph} \leftarrow \mathsf{Enc}(w, I_{\mathsf{Enc}})$ and $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$, then $1 \leftarrow \mathsf{Search}(\mathsf{cph}, \mathsf{tk})$ always holds.

Similar to the searchable encryption in the literature, the intuition for ABKS security is that the adversary should learn nothing but search results when executing function Search. Concretely, an ABKS scheme should satisfy security requirements as follows even in the presence of a probabilistic polynomial-time adversary $\mathcal{A}$:

- Selective security against chosen-keyword attack: Given only keyword ciphertexts (i.e. without seeing any matched search tokens), no partial information about keywords is leaked in the selective model (meaning that I_{Enc} for encrypting challenge keywords has to be selected before the public parameter is generated). To be specific,

$\mathcal{A}$ cannot distinguish the encryption of two challenge keywords of its choice. We formalize this security property via the selectively chosen-keyword attack game.

- Keyword secrecy: In the public-key setting, it is inherently impossible to protect any information encoded by search tokens (aka. predicate privacy [36]) due to the *keyword guessing attack*: one can encrypt any keyword of his choice and check whether the resulting ciphertext matches the target token. Therefore, a weaker notion, called *keyword secrecy*, is introduced to assure that the probability of $\mathcal{A}$ learning the keyword from the ciphertext and its search token is no more than that of a random keyword guess. We formalize this security property via the keyword secrecy game.

Selectively Chosen-Keyword Attack (SCKA) Game:

Setup: $\mathcal{A}$ selects a non-trivial challenge I^*_{Enc} (a trivial challenge I^*_{Enc} is one that can be satisfied by any data user who does not have any credential), and gives it to the challenger. Then the challenger runs $\mathsf{Setup}(1^\ell)$ to generate the public parameter pm and the master key mk.

Phase 1: $\mathcal{A}$ can query the following oracles for polynomially many times, and the challenger keeps a keyword list L_{kw}, which is initially empty.

- $\mathcal{O}_{\mathsf{KeyGen}}(I_{\mathsf{KeyGen}})$: If $F(I_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$, then abort; otherwise, the challenger returns to $\mathcal{A}$ credential sk corresponding to I_{KeyGen}.
- $\mathcal{O}_{\mathsf{TokenGen}}(I_{\mathsf{KeyGen}}, w)$: The challenger generates credential sk with I_{KeyGen}, and returns to $\mathcal{A}$ a search token tk by running algorithm $\mathsf{TokenGen}$ with inputs sk and w. If $F(I_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$, the challenger adds w to L_{kw}.

Challenge phase: $\mathcal{A}$ chooses two keywords w_0 and w_1, where $w_0, w_1 \notin L_{kw}$. The challenger selects $\lambda \leftarrow \{0,1\}$, computes $\mathsf{cph}^* \leftarrow \mathsf{Enc}(w_\lambda, I^*_{\mathsf{Enc}})$, and delivers cph^* to $\mathcal{A}$. Note that the requirement of $w_0, w_1 \notin L_{kw}$ is to prevent $\mathcal{A}$ from trivially guessing λ with tokens from $\mathcal{O}_{\mathsf{TokenGen}}$.

Phase 2: $\mathcal{A}$ continues to query the oracles as in Phase 1. The restriction is that $(I_{\mathsf{KeyGen}}, w_0)$ and $(I_{\mathsf{KeyGen}}, w_1)$ cannot be the input to $\mathcal{O}_{\mathsf{TokenGen}}$ if $F(I_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$.

Guess: $\mathcal{A}$ outputs a bit λ', and wins the game if $\lambda' = \lambda$.

Let $|\mathsf{Pr}[\lambda = \lambda'] - \frac{1}{2}|$ be the advantage of $\mathcal{A}$ winning the above SCKA game. Thus, we have

Definition 2: An ABKS scheme is *selectively secure against chosen-keyword attack* if the advantage of any $\mathcal{A}$ winning the SCKA game is negligible in security parameter ℓ.

Keyword Secrecy Game:

Setup: The challenger runs $\mathsf{Setup}(1^\ell)$ to generate the public parameter pm and the master key mk.

Phase 1: $\mathcal{A}$ can query the following oracles for polynomially many times:

- $\mathcal{O}_{\mathsf{KeyGen}}(I_{\mathsf{KeyGen}})$: The challenger returns to $\mathcal{A}$ credential sk corresponding to I_{KeyGen}. It adds I_{KeyGen} to the list L_{KeyGen}, which is initially empty.
- $\mathcal{O}_{\mathsf{TokenGen}}(I_{\mathsf{KeyGen}}, w)$: The challenger generates credential sk with I_{KeyGen}, and returns to $\mathcal{A}$ a search token tk by running algorithm $\mathsf{TokenGen}$ with input sk and w.

Challenge phase: $\mathcal{A}$ chooses a non-trivial I^*_{Enc} and gives it to the challenger. The challenger selects w^* from the message space uniformly at random and selects I^*_{KeyGen} such that $F(I^*_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$. The challenger runs $\mathsf{cph} \leftarrow \mathsf{Enc}(w^*, I^*_{\mathsf{Enc}})$ and $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w^*)$ and delivers $(\mathsf{cph}, \mathsf{tk})$ to $\mathcal{A}$. We require that $\forall I_{\mathsf{KeyGen}} \in L_{\mathsf{KeyGen}}$, $F(I_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 0$.

Guess: After guessing q distinct keywords, $\mathcal{A}$ outputs a keyword w', and wins the game if $w' = w$.

Definition 3: An ABKS scheme achieves *keyword secrecy* if the probability that $\mathcal{A}$ wins the keyword secrecy game is at most $\frac{1}{|\mathcal{M}|-q} + \epsilon$, where $\mathcal{M}$ is the keyword space, q is the number of distinct keywords that the adversary has attempted, and ϵ is a negligible in security parameter ℓ.

B. ABKS *Construction*

Assume that $H_1 : \{0,1\}^* \to G$ is a secure hash function that modeled as random oracle, and $H_2 : \{0,1\}^* \to \mathbb{Z}_p$ is a secure one-way hash function, where G is the cyclic group of bilinear map $e : G \times G \to G_T$.

In ABKS it should consider (1) how to encrypt a keyword, and (2) how to generate a search token from credentials and a keyword, so that if data users' attributes (or credentials) satisfy the specified access control policy, then it can determine whether keywords are equal with respect to the ciphertext and search token.

By taking KP-ABKS as an example, where credentials are generated by letting $t \leftarrow \mathbb{Z}_p$, $A_v = g^{q_v(0)} H_1(\mathsf{att}(v))^t, B_v = g^t$ for each leaf v, where g is the generator of G, $q_v(0)$ is the share of secret ac for leaf v according to the access tree T, the ciphertext and search token are generated as follows:

- Keyword w is encrypted into two parts: one is to blind keyword with randomness by letting $W' = g^{cr_1}$, $W = g^{a(r_1+r_2)} g^{bH_2(w)r_1}$ and $W_0 = g^{r_2}$ where $g^a, g^b, g^c \in G$ are public keys and $r_1, r_2 \leftarrow \mathbb{Z}_p$, and the other is associated with the attribute set Atts by letting $W_j = H_1(\mathsf{at}_j)^{r_2}$ for each $\mathsf{at}_j \in \mathsf{Atts}$, both of which are binding together with r_2.
- Given credentials, a search token for keyword w can be generated with two parts: one is associated with keywords, $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s$ and $\mathsf{tok}_2 = g^{cs}$ by selecting $s \leftarrow \mathbb{Z}_p$, and the other is associated with credentials by letting $A'_v = A^s_v, B'_v = B^s_v$ for each $v \in \mathsf{lvs}(\mathsf{T})$, both of which are binding together with the random number s.

As long as the attribute set satisfies the access tree T, with A'_v, B'_v and W_0, W_j, it can recover $e(g,g)^{acr_2 s}$, which can be used to test the keyword equality for the keyword ciphertext and search token, the detail of which is described in the following KP-ABKS construction.

KP-ABKS **Construction.** The KP-ABKS can be constructed as follows:

$\mathsf{Setup}(1^\ell)$: This algorithm selects a bilinear map $e : G \times G \to G_T$, such that G and G_T are cyclic groups of order p, an ℓ-bit prime. It selects $a, b, c \leftarrow \mathbb{Z}_p$ and $g \leftarrow G$. Let

$$\mathsf{pm} = (e, g, p, g^a, g^b, g^c, G, G_T), \mathsf{mk} = (a, b, c).$$

$\mathsf{KeyGen}(\mathsf{mk}, \mathsf{T})$: This algorithm executes $\{q_v(0)|v \in \mathsf{lvs}(\mathsf{T})\} \leftarrow \mathsf{Share}(\mathsf{T}, ac)$ to obtain secret shares of ac for all leaves in T. For each leaf v in T, it picks $t \leftarrow \mathbb{Z}_p$, and computes $A_v = g^{q_v(0)} H_1(\mathsf{att}(v))^t$ and $B_v = g^t$. Let

$$\mathsf{sk} = (\mathsf{T}, \{(A_v, B_v)|v \in \mathsf{lvs}(\mathsf{T})\}).$$

$\mathsf{Enc}(w, \mathsf{Atts})$: This algorithm selects $r_1, r_2 \leftarrow \mathbb{Z}_p$, and computes $W' = g^{cr_1}$, $W = g^{a(r_1+r_2)} g^{bH_2(w)r_1}$ and $W_0 = g^{r_2}$. For each $\mathsf{at}_j \in \mathsf{Atts}$, it computes $W_j = H_1(\mathsf{at}_j)^{r_2}$. Let

$$\mathsf{cph} = (\mathsf{Atts}, W', W, W_0, \{W_j|\mathsf{at}_j \in \mathsf{Atts}\}).$$

$\mathsf{TokenGen}(\mathsf{sk}, w)$: This algorithm selects $s \leftarrow \mathbb{Z}_p$, and computes $A'_v = A_v^s, B'_v = B_v^s$ for each $v \in \mathsf{lvs}(\mathsf{T})$. It computes $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s$ and $\mathsf{tok}_2 = g^{cs}$. Let

$$\mathsf{tk} = (\mathsf{tok}_1, \mathsf{tok}_2, \mathsf{T}, \{(A'_v, B'_v)|v \in \mathsf{lvs}(\mathsf{T})\})$$

$\mathsf{Search}(\mathsf{tk}, \mathsf{cph})$: Given the attribute set Atts included in cph, this algorithm selects a minimum attribute set S satisfying the access tree T included in tk. If S does not exist, return 0; otherwise, for each $\mathsf{at}_j \in S$, it computes

$$E_v = e(A'_v, W_0)/e(B'_v, W_j) = e(g, g)^{sr_2 q_v(0)},$$

where $\mathsf{att}(v) = \mathsf{at}_j$ for $v \in \mathsf{lvs}(\mathsf{T})$. Then it computes

$$e(g, g)^{sr_2 q_{\mathsf{root}}(0)} \leftarrow \mathsf{Combine}(\mathsf{T}, \{E_v|\mathsf{att}(v) \in S\})$$

so that $E_{\mathsf{root}} = e(g, g)^{acsr_2}$. It return 1 if $e(W', \mathsf{tok}_1) E_{\mathsf{root}} = e(W, \mathsf{tok}_2)$; otherwise return 0.

This completes the description of KP-ABKS scheme. The correctness can be verified as follows.

$$\begin{aligned} e(W', \mathsf{tok}_1) E_{\mathsf{root}} &= e(g^{cr_1}, (g^a g^{bH_2(w)})^s) E_{\mathsf{root}} \\ &= ee(g, g)^{acsr_1} e(g, g)^{bcsH_2(w)r_1} E_{\mathsf{root}} \\ &= e(g, g)^{acs(r_1+r_2)} e(g, g)^{bcsH_2(w)r_1} \\ and \quad e(W, \mathsf{tok}_2) &= ee(g^{a(r_1+r_2)} g^{bH_2(w)r_1}, g^{cs}) \\ &= ee(g, g)^{acs(r_1+r_2)} e(g, g)^{bcsH_2(w)r_1} \end{aligned}$$

To show the security of KP-ABKS, we have the following theorems, the proofs of which are deferred to Appendix A and Appendix B, respectively.

Theorem 1: Assume that the DL assumption holds, the KP-ABKS scheme is selectively secure against chosen-keyword attack in the random oracle model.

Theorem 2: Given the one-way hash function H_2, the KP-ABKS scheme achieves keyword secrecy.

CP-ABKS Construction. The CP-ABKS scheme can be constructed as below:

$\mathsf{Setup}(1^\ell)$: This algorithm selects a bilinear group $e : G \times G \rightarrow G_T$, such that G and G_T are cyclic groups of order p, an $\ell-$bit prime. It selects $a, b, c \leftarrow \mathbb{Z}_p$ and $g \leftarrow G$. Let

$$\mathsf{pm} = (e, g, p, g^a, g^b, g^c, G, G_T), \mathsf{mk} = (a, b, c).$$

$\mathsf{KeyGen}(\mathsf{mk}, \mathsf{Atts})$: This algorithm selects $r \leftarrow \mathbb{Z}_p$, computes $A = g^{(ac-r)/b}$. Then for each $\mathsf{at}_j \in \mathsf{Atts}$, it selects $r_j \leftarrow \mathbb{Z}_p$ and computes $A_j = g^r H_1(\mathsf{at}_j)^{r_j}$ and $B_j = g^{r_j}$. Let

$$\mathsf{sk} = (\mathsf{Atts}, A, \{(A_j, B_j)|\mathsf{at}_j \in \mathsf{Atts}\}).$$

$\mathsf{Enc}(w, \mathsf{T})$: This algorithm selects $r_1, r_2 \leftarrow \mathbb{Z}_p$, and computes $W = g^{cr_1}$, $W_0 = g^{a(r_1+r_2)} g^{bH_2(w)r_1}$ and $W' = g^{br_2}$. It further computes secret shares of r_2 for all leaves of T by running $\{q_v(0)|v \in \mathsf{lvs}(\mathsf{T})\} \leftarrow \mathsf{Share}(\mathsf{T}, r_2)$. Then for each $v \in \mathsf{lvs}(\mathsf{T})$, it computes $W_v = g^{q_v(0)}$ and $D_v = H_1(\mathsf{att}(v))^{q_v(0)}$. Let

$$\mathsf{cph} = (\mathsf{T}, W, W_0, W', \{(W_v, D_v)|v \in \mathsf{lvs}(\mathsf{T})\})$$

$\mathsf{TokenGen}(\mathsf{sk}, w)$: This algorithm selects $s \leftarrow \mathbb{Z}_p$, and computes $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s, \mathsf{tok}_2 = g^{cs}$ and $\mathsf{tok}_3 = A^s = g^{(acs-rs)/b}$. Then for each $\mathsf{at}_j \in \mathsf{Atts}$, it computes $A'_j = A_j^s$ and $B'_j = B_j^s$. Let

$$\mathsf{tk} = (\mathsf{Atts}, \mathsf{tok}_1, \mathsf{tok}_2, \mathsf{tok}_3, \{(A'_j, B'_j)|\mathsf{at}_j \in \mathsf{Atts}\}).$$

$\mathsf{Search}(\mathsf{tk}, \mathsf{cph})$: Given an attribute set Atts included in tk, this algorithm selects a minimum attribute set S that satisfies the access tree T included in cph. If S does not exist, return 0; otherwise, for each $\mathsf{at}_j \in S$, it computes

$$E_v = e(A'_j, W_v)/e(B'_j, D_v) = e(g, g)^{rsq_v(0)},$$

where $\mathsf{att}(v) = \mathsf{at}_j$ for $v \in \mathsf{lvs}(\mathsf{T})$. Then it computes

$$e(g, g)^{rsq_{\mathsf{root}}(0)} \leftarrow \mathsf{Combine}(\mathsf{T}, \{E_v|\mathsf{att}(v) \in S\})$$

so that $E_{\mathsf{root}} = e(g, g)^{rsr_2}$. It returns 1 if $e(W_0, \mathsf{tok}_2) = e(W, \mathsf{tok}_1) E_{\mathsf{root}} e(\mathsf{tok}_3, W')$; otherwise return 0.

This completes the CP-ABKS construction, and its correctness can be verified similar to that of KP-ABKS. To show the security of CP-ABKS, we have Theorem 3 and Theorem 4. The proof of former one is deferred to Appendix C, and the proof of latter one is omitted because it is similar to that of Theorem 2.

Theorem 3: CP-ABKS scheme is selectively secure against chosen-keyword attack in the generic bilinear group model.

Theorem 4: Given the one way hash function H_2, the CP-ABKS scheme achieves keyword secrecy.

IV. Verifiable Attribute-Based Keyword Search

While ABKS enables data owners to grant search capability over outsourced encrypted keywords, it is not adequate for secure data sharing in the cloud computing environment in that the cloud is untrusted, meaning that the cloud might return incomplete/manipulated search result. Therefore, in order to combat this, it requires another layer protection – verifiability, validating whether the cloud has faithfully executed search operations and returned research result honestly.

A. System Model and Threat Model

We consider the system model as illustrated in Figure 1, which is involved four entities: the data owner, who has large amount of data indexed by keywords (i.e. inverted index as in Figure 1) and will store it in the cloud; the cloud, which is to provide data storage and retrieval service and has sufficient storage space and computation resource; the data user, who is to retrieve data owner's encrypted data according to some keyword; and the trusted authority that issues credentials and necessary public/private keys to all

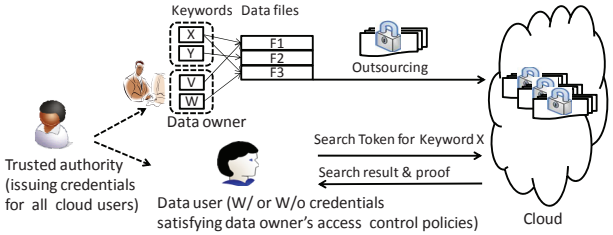


Fig. 1. System model for verifiable attribute-based keyword search over outsourced encrypted data, where keywords X, Y and V, W have different access control policies.

cloud users. We assume that the authentication between any two entities can be appropriately done.

The data owner encrypts its own data (i.e. keywords and their corresponding data files) before outsourcing for the sake of data privacy. The data owner can specify access control policies to determine who can generate appropriate search token to conduct search over encrypted keywords. When the data user wishes to retrieve encrypted data (i.e. encrypted keyword and the associated encrypted data files) from the cloud according to some keyword, he generates a search token and delivers it to the cloud. Upon request, the cloud conducts keyword search over encrypted data and returns the corresponding encrypted data file to the user. To show its faithful execution, the cloud also returns necessary proof so that the user can verify that the returned result is correct.

We assume that the data owner are trusted, and data users (i.e. w/ or w/o proper attributes satisfying the data owner's access control policies) are semi-trusted, meaning they may try to derive other sensitive information beyond their own. We assume that the cloud server is untrusted: he might either conduct search operation with arbitrary input, or return incorrect results which were manipulated.

B. VABKS *Definition and Security*

Let $\mathsf{D} = (\mathsf{KS} = \{\mathsf{KS}_1, \ldots, \mathsf{KS}_l\}, \mathsf{MP} = \{\mathsf{MP}(w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}, \mathsf{FS} = \{\mathsf{F}_1, \ldots, \mathsf{F}_n\})$ denote the inverted index and the set of data files, where $\mathsf{KS}_i, 1 \leq i \leq l$, is the set of keywords that will be encrypted with the same access control policy, $\mathsf{F}_j, 1 \leq j \leq n$, is the data file, and $\mathsf{MP}(w)$ denote the set of ordered identifiers identifying data files associated with keyword w.

Definition 4: A VABKS scheme consists of the following algorithms:

- $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Init}(1^\ell)$: This algorithm is run by a trusted party (e.g., attribute authority) when initializing the system.
- $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: This algorithm is run by the trusted party to issue credential sk to a data user.
- $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}) \leftarrow \mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$: This algorithm is run by the data owner to encrypt data collection $\mathsf{D} = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS})$ and obtain encrypted data files $\mathsf{D}_{\mathsf{cph}}$, encrypted index Index and auxiliary information Au, where $\{I_{\mathsf{Enc}}\}_l$ is the set of access control policies to encrypt KS and $\{I'_{\mathsf{Enc}}\}_n$ is the set of access control policies to encrypt FS.
- $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$: This algorithm is run by the data user, who has the credential sk, to issue search token tk on keyword w.
- $(\mathsf{proof}, \mathsf{rslt}) \leftarrow \mathsf{SearchIndex}(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}, \mathsf{tk})$: This algorithm is run by the cloud to perform keyword search over encrypted data on behalf of the data user. It outputs the search result rslt and a proof proof to show that the result is correct.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{sk}, w, \mathsf{tk}, \mathsf{rslt}, \mathsf{proof})$: This algorithm is run by the data user to verify that $(\mathsf{rslt}, \mathsf{proof})$ is valid with respect to search token tk.

The correctness of the VABKS scheme requires that, given $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Init}(1^\ell)$, $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$, for any keyword-based data collection D and keyword w, $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}) \leftarrow \mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$, $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$ and $(\mathsf{proof}, \mathsf{rslt}) \leftarrow \mathsf{SearchIndex}(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}, \mathsf{tk})$, then $1 \leftarrow \mathsf{Verify}(\mathsf{sk}, w, \mathsf{tk}, \mathsf{rslt}, \mathsf{proof})$ always holds.

Intuitively, a VABKS scheme should satisfy the following security requirements even in the presence of a probabilistic polynomial-time adversary $\mathcal{A}$ (i.e. the cloud server):

- Data secrecy: Even if $\mathcal{A}$ is given keywords, it still cannot learn any information about encrypted data files. It can be formalized via the conventional chosen-plaintext security game with additional requirement that the two challenge data $\mathsf{D}_0 = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}_0), \mathsf{D}_1 = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}_1)$ have the same KS and MP, and $|\mathsf{FS}_0| = |\mathsf{FS}_1|$. Otherwise, $\mathcal{A}$ can distinguish FS_0 and FS_1 by observing the search result with the search token for a specific keyword.
- Selective security against chosen-keyword attack: Without seeing corresponding search tokens, $\mathcal{A}$ cannot guess which of the given keyword sets was encrypted. It can be extended from the definition of selective security against chosen-keyword attack of ABKS.
- Keyword secrecy: Even if $\mathcal{A}$ is given the data, the probability of $\mathcal{A}$ learning the keyword from the keyword ciphertext and the search token is no more than of a random guess within the set of keywords which composes the given data sets. This security definition also can be extended from the keyword secrecy of ABKS.
- Verifiability: $\mathcal{A}$ cannot return incorrect search results without being caught. Specifically, after executing algorithm $\mathsf{SearchIndex}$, $\mathcal{A}$ responses with the search result and the proof. If $\mathcal{A}$ returns incorrect and/or incomplete search result, the cheating behavior can be detected by algorithm Verify with overwhelming probability. We formalize this security property via the following verifiability game.

Verifiability Game:

Setup: The challenger runs $(\mathsf{pm}, \mathsf{mk}) \leftarrow \mathsf{Init}(1^\ell)$. $\mathcal{A}$ selects $\mathsf{D} = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}), \{I_{\mathsf{Enc}}\}_l$ and $\{I'_{\mathsf{Enc}}\}_n$ and sends them to the challenger. The challenger runs $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}) \leftarrow \mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$, and gives $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}})$ to $\mathcal{A}$.

Phase 1: $\mathcal{A}$ can query the following oracles for polynomially many times.

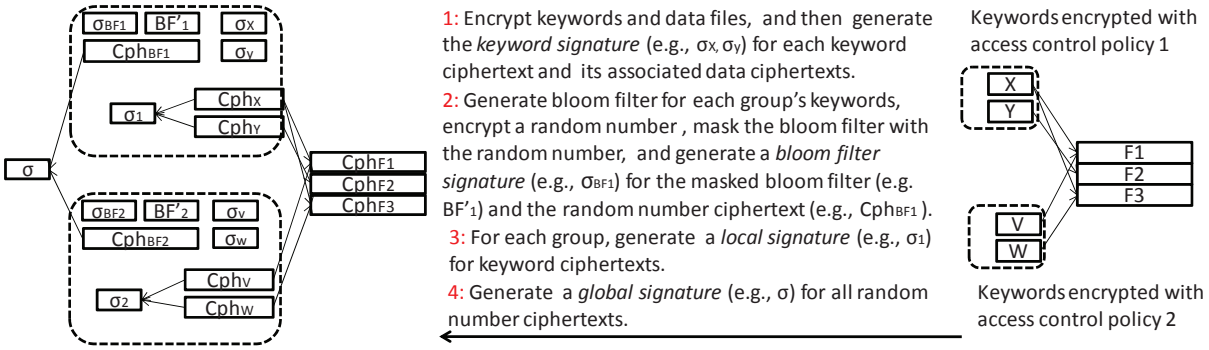


Fig. 2. Basic idea for verifiability, where data files F_1, F_2, F_3 were encrypted to $\mathsf{cph}_{F_1}, \mathsf{cph}_{F_2}, \mathsf{cph}_{F_3}$, keywords X, Y were encrypted to $\mathsf{cph}_X, \mathsf{cph}_Y$ with access control policy 1, and keywords V, W were encrypted to $\mathsf{cph}_V, \mathsf{cph}_W$ with access control policy 2 (different from access control policy 1).

- $\mathcal{O}_{\mathsf{KeyGen}}(I_{\mathsf{KeyGen}})$: The challenger returns to $\mathcal{A}$ credential sk corresponding to I_{KeyGen}.
- $\mathcal{O}_{\mathsf{TokenGen}}(I_{\mathsf{KeyGen}}, w)$: The challenger generates credential sk with I_{KeyGen}, and returns to $\mathcal{A}$ a search token tk by running algorithm $\mathsf{TokenGen}$ with inputs sk and w.
- $\mathcal{O}_{\mathsf{Verify}}(I_{\mathsf{KeyGen}}, w, \mathsf{tk}, \mathsf{rslt}, \mathsf{proof})$: The challenger generates credential sk with I_{KeyGen}, returns γ to $\mathcal{A}$ by running $\gamma \leftarrow \mathsf{Verify}(\mathsf{sk}, w, \mathsf{tk}, \mathsf{rslt}, \mathsf{proof})$.

Challenge phase: $\mathcal{A}$ selects a non-trivial challenge I^*_{Enc} and a keyword w^* and gives them to the challenger. The challenger selects I^*_{KeyGen} such that $F(I^*_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$, generates credential sk^* with I^*_{KeyGen} and returns to $\mathcal{A}$ a search token tk^* by running $\mathsf{tk}^* \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w^*)$.

Guess: $\mathcal{A}$ outputs $(\mathsf{rslt}^*, \mathsf{proof}^*)$ to the challenger. We say $\mathcal{A}$ wins the game if $1 \leftarrow \mathsf{Verify}(\mathsf{sk}^*, w^*, \mathsf{tk}^*, \mathsf{rslt}^*, \mathsf{proof}^*)$ and $\mathsf{rslt}^* \neq \mathsf{rslt}$, where $(\mathsf{rslt}, \mathsf{proof})$ is produced by the challenger by running $\mathsf{SearchIndex}(\mathsf{Au}, \mathsf{Index}, \mathsf{tk}^*)$.

Definition 5: A VABKS scheme is verifiable if the advantage that any $\mathcal{A}$ wins the verifiability game is negligible in security parameter ℓ.

C. VABKS Construction

It is natural to adopt ABKS as a building block to enable keyword search over encrypted data and grant search capability, and let us focus on how to achieve verifiability for VABKS.

Note that when keywords are to be encrypted, they are separated into multiple groups according to access control policies, so that keywords within each group are unique (meaning non redundant keyword exists in the same group). To achieve verifiability, the data user should verify that (1) the cloud performed search operations on all group's keyword ciphertexts; and (2) the cloud honestly returned the search result for each group, which could be either a null result or one keyword ciphertext and its associated data ciphertexts.

Specifically, we elaborate the basic idea for achieving verifiability in Figure 2, where the data owner generates the signatures and bloom filters as follows:

- The first type of signature, called *keyword signature*, is generated for each keyword ciphertext and its associated data ciphertexts, which is prevent the cloud from returning part of data ciphertexts as the search result.
- For each group, one bloom filter is built from its keywords, which enables data users to *efficiently* check that the queried keyword was indeed not in the group when the cloud returns a null result, without downloading all keyword ciphertexts from the cloud. A random number is selected and encrypted with the same access control policy as keywords, and is used to mask the bloom filter for preserving keyword privacy. The second type of signature, called *bloom filter signature*, is generated for the masked bloom filter and the random number ciphertext for assuring their integrity.
- The third type of signature, called *global signature*, is obtained by signing random number ciphertexts of all groups. It enforces the cloud to perform search operations over all groups' keyword ciphertexts. The reason of selecting random number ciphertext is that it has identical access control policy with that of keyword ciphertext within the same group, and can be used to assure that the cloud only return one result (i.e. null or one keyword ciphertext and its associated ciphertexts) for each group.
- The last type of signature, called *local signature*, is generated from all keyword ciphertexts within the same group. When data users download one group's keyword ciphertexts from the cloud (due to the false positive property of the bloom filter), this signature is to validate the integrity of those keyword ciphertexts.

The VABKS scheme is depicted in Figure 3, which makes use of a symmetric encryption $\mathsf{SE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, an attribute-based encryption scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, both of which are used to encrypt data files, an attribute-based keyword search scheme $\mathsf{ABKS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{TokenGen}, \mathsf{Search})$, which is to encrypt keywords, and a signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. Here ABE and ABKS are either ciphertext-policy or key-policy, so that we have two variants of VABKS.

In algorithm Verify of Figure 3, when the authorized data user verifies a null result for the group $\{\mathsf{cph}_w | w \in \mathsf{KS}_i\}$, given a search token for keyword w', it can happen that $1 \leftarrow \mathsf{BFVerify}(\{H'_1, \ldots, H'_k\}, \mathsf{BF}_i, w')$ but the matched keyword ciphertext was not stored in the cloud. The probability

$\mathsf{Init}(1^\ell)$: Given security parameter ℓ, the attribute authority chooses k hash functions $H'_1, \ldots, H'_k$, which are used to construct m-bit Bloom filter. Let $H : \{0,1\}^\ell \rightarrow \{0,1\}^m$ be a secure pseudorandom generator. It executes $(\mathsf{ABE.pm}, \mathsf{ABE.mk}) \leftarrow \mathsf{ABE.Setup}(1^\ell)$ and $(\mathsf{ABKS.pm}, \mathsf{ABKS.mk}) \leftarrow \mathsf{ABKS.Setup}(1^\ell)$. Let $\mathsf{pm} = (\mathsf{ABE.pm}, \mathsf{ABKS.pm}, H'_1, \ldots, H'_k)$ which is public known, and $\mathsf{mk} = (\mathsf{ABE.mk}, \mathsf{ABKS.mk})$.

$\mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: The attribute authority runs $\mathsf{ABE.sk} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mk}, I_{\mathsf{KeyGen}})$ and $\mathsf{ABKS.sk} \leftarrow \mathsf{ABKS.KeyGen}(\mathsf{ABKS.mk}, I_{\mathsf{KeyGen}})$, and sets $\mathsf{sk} = (\mathsf{ABE.sk}, \mathsf{ABKS.sk})$, which is delivered to the user via authenticated private channel.

$\mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$: If the data owner has no public/private keys for the signature scheme Sig, then it runs $(\mathsf{Sig.sk}, \mathsf{Sig.pk}) \leftarrow \mathsf{Sig.KeyGen}(1^\ell)$, by keeping Sig.sk private and making Sig.pk public. Given $\mathsf{D} = (\mathsf{KS} = \{\mathsf{KS}_1, \ldots, \mathsf{KS}_l\}, \mathsf{MP} = \{\mathsf{MP}(w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}, \mathsf{FS} = \{\mathsf{F}_1, \ldots, \mathsf{F}_n\})$, the data owner executes as follows:

1) Encrypt the data file with hybrid encryption: $\forall \mathsf{F}_j \in \mathsf{FS}$, it generates the ciphertext $\mathsf{cph}_{\mathsf{F}_j} = (\mathsf{cph}_{\mathsf{sk}_j}, \mathsf{cph}_{\mathsf{SE}_j})$ by running $\mathsf{SE.sk}_j \leftarrow \mathsf{SE.KeyGen}()$, $\mathsf{cph}_{\mathsf{SE}_j} \leftarrow \mathsf{SE.Enc}(\mathsf{SE.sk}_j, \mathsf{F}_j)$, and $\mathsf{cph}_{\mathsf{sk}_j} \leftarrow \mathsf{ABE.Enc}(I'_{\mathsf{Enc}j}, \mathsf{SE.sk}_j)$.
2) Encrypt each keyword and generate keyword signature: Given $\mathsf{KS}_i, 1 \leq i \leq l$, for each $w \in \mathsf{KS}_i$, it runs $\mathsf{cph}_w \leftarrow \mathsf{ABKS.Enc}(I_{\mathsf{Enc}_i}, w)$, sets $\mathsf{MP}(\mathsf{cph}_w) = \{\mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} | \mathsf{ID}_{\mathsf{F}_j} \in \mathsf{MP}(w)\}$, and generates $\sigma_w \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{cph}_w || \{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\})$, where identities in $\mathsf{MP}(\mathsf{cph}_w)$ are ordered, and $\mathsf{ID}_{\mathsf{F}_j}$ and $\mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}}$ are identities identifying data file F_j and data ciphertext $\mathsf{cph}_{\mathsf{F}_j}$, respectively.
3) Generate the bloom filter, bloom filter signature and local signature for each group KS_i: Let $\mathsf{BF}_i \leftarrow \mathsf{BFGen}(\{H'_1, \ldots, H'_k\}, \mathsf{KS}_i)$, $\mathsf{cph}_{\mathsf{BF}_i} \leftarrow \mathsf{ABE.Enc}(I_{\mathsf{Enc}_i}, \mathsf{M})$ by randomly selecting M from the message space of ABE, compute $\mathsf{BF}'_i = H(\mathsf{M}) \bigotimes \mathsf{BF}_i$ and generate $\sigma_{\mathsf{BF}_i} \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{BF}' || \mathsf{cph}_{\mathsf{BF}_i})$. Let $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{cph}_{w_1} || \ldots || \mathsf{cph}_{|\mathsf{KS}_i|})$.
4) Generate the global signature: Let $\sigma = \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{cph}_{\mathsf{BF}_1} || \ldots || \mathsf{cph}_{\mathsf{BF}_l})$.
5) Let $\mathsf{Au} = (\sigma, \sigma_1, \ldots, \sigma_l, \sigma_{\mathsf{BF}_1}, \ldots, \sigma_{\mathsf{BF}_l}, \{\sigma_w | w \in \cup_{i=1}^{l} \mathsf{KS}_i\})$, $\mathsf{Index} = (\{\mathsf{cph}_w | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}, \{\mathsf{MP}(\mathsf{cph}_w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\})$ and $\mathsf{D}_{\mathsf{cph}} = (\{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{F}_j \in \mathsf{FS}\})$.

$\mathsf{TokenGen}(\mathsf{sk}, w)$: Given sk, the data user generates search token $\mathsf{tk} \leftarrow \mathsf{ABKS.TokenGen}(\mathsf{ABKS.sk}, w)$.

$\mathsf{SearchIndex}(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}, \mathsf{tk})$: Let rslt be an empty set and $\mathsf{proof} = (\sigma)$ initially. Let $\prod = \{\mathsf{cph}_w | w \in \mathsf{KS}_i\}$ be the set of keyword ciphertexts having the identical access control policy. For each $\mathsf{cph}_w \in \prod$, it runs $\gamma \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$, and if $\gamma = 1$, then adds $(\mathsf{cph}_w, \{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\})$ to rslt and $(\sigma_w, \mathsf{cph}_{\mathsf{BF}_i})$ to proof. If there exist no $\gamma = 1$ after enumerating all cph_w in S, then add $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i})$ to the proof.

$\mathsf{Verify}(\mathsf{sk}, w, \mathsf{tk}, \mathsf{proof}, \mathsf{rslt})$: The data user verifies the search result from the cloud as follows:

1) Verify the integrity of random number ciphertexts: $\gamma = \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma, \mathsf{cph}_{\mathsf{BF}_1} || \ldots || \mathsf{cph}_{\mathsf{BF}_l})$. If $\gamma = 0$, then returns 0.
2) For $i = 1, \ldots, l$, it executes as follows to ensure that the cloud returned correct result for each group:
   Case 1: When the search result with respect to the group is not null, meaning $(\mathsf{cph}_w, \{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\})$ is included in rslt, then it runs $\gamma \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$ and $\gamma' \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_w, \mathsf{cph}_w || \{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\})$. If either $\gamma = 0$ or $\gamma' = 0$, then return 0.
   Case 2: When the search result with respect to the group is null, meaning $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i})$ is included in rslt, then it runs $\gamma' \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_{\mathsf{BF}_i}, \mathsf{BF}'_i || \mathsf{cph}_{\mathsf{BF}_i})$. If $\gamma' = 0$, return 0; otherwise, it executes as follows:
   - If the data user is authorized, it computes $\mathsf{M} \leftarrow \mathsf{ABE.Dec}(\mathsf{ABE.sk}, \mathsf{cph}_{\mathsf{BF}_i})$, $\mathsf{BF}_i = H(\mathsf{M}) \bigotimes \mathsf{BF}'_i$, and $\delta \leftarrow \mathsf{BFVerify}(\{H'_1, \ldots, H'_k\}, \mathsf{BF}, w)$. If $\delta = 1$, then it downloads $\{\mathsf{cph}_w | w \in \mathsf{KS}_i\}$ and σ_i from the cloud, and runs $\eta \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_i, \mathsf{cph}_{w_1} || \ldots || \mathsf{cph}_{w_{|\mathsf{KS}_i|}})$. If $\eta = 0$, return 0; otherwise it runs $\tau \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$ for each cph_w. If there exists some $\tau = 1$, return 0.
   - If the data user's attributes do not satisfy the access control policy, then he continues because $\mathsf{cph}_{\mathsf{BF}_i}$ cannot be decrypted.
3) Return 1, meaning that the search result rslt from the cloud is correct.

Fig. 3. VABKS construction

that this event happens equals to the false-positive rate of the Bloom filter. In order to validate the search result in this case, algorithm Verify has to download $\{\mathsf{cph}_w | w \in \mathsf{KS}_i\}$, and check one by one. Because this incurs high computational and communication cost, we can set the false-positive rate as low as possible by choosing appropriate m and k. For example, in our experiment we set the false-positive rate to be 4.5×10^{-9}.

D. Security Analysis

We show that the proposed VABKS scheme satisfies the security requirements with the following theorems, the formal proofs of which are deferred to Appendix D.

We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking VABKS's data secrecy, then it breaks the assumption that CPA-secure ABEand CPA-secure SE. This is formally achieved via Theorem 5.

Theorem 5: If the ABE is CPA-secure and SE is CPA-secure, the VABKS scheme achieves data secrecy.

We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking VABKS's selective security against chosen keyword attack, then it breaks the assumption that ABKS achieves the selective security against chosen keyword attack, given that ABE is CPA-secure and H is a secure pseudorandom generator. This is formally achieved via Theorem 6.

Theorem 6: If ABE is CPA-secure, H is a secure pseudorandom generator and the ABKS achieves selective security against chosen keyword attack, the VABKS scheme also achieves security against chosen keyword attack.

We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking keyword secrecy of the VABKS, then it breaks the assumption that ABKS achieves keyword secrecy, given that ABE is CPA-secure and H is a secure pseudorandom generator. This is formally achieved via Theorem 7.

Theorem 7: If ABE is CPA-secure, H is a secure pseudorandom generator and the ABKS achieves keyword secrecy, the VABKS scheme also achieves keyword secrecy.

We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking verifiability of the VABKS then it breaks the Sig's unforgeability. This is formally achieved via Theorem 8.

Theorem 8: If Sig is a secure signature, the VABKS construction achieves the verifiability.

V. Performance Evaluation

We evaluate efficiency of the ABKS schemes in terms of both asymptotic complexity and actual execution time, and efficiency of the VABKS scheme in terms of actual execution time. Asymptotic complexity is measured in terms of four kinds of operations: H_1 denotes the operation of mapping a bit-string to an element of G, Pair denotes the pairing operation, E denotes the exponentiation operation in G, and E_T denotes the exponentiation operation in G_T. We ignore multiplication and hash operations (other than H_1) because they are much more efficient than the above operations (see benchmark of [37]).

We implemented ABKS and VABKS in the JAVA language based on the Java Pairing Based Cryptography library (jPBC) [37]. In our implementations, we instantiated the bilinear map with Type A pairing, which offers a level of security that is equivalent to 1024-bit DLOG [37]. For both CP-VABKS and KP-VABKS, we instantiated the symmetric encryption scheme with AES, and the signature scheme with DSA signature scheme provided by JDK1.6. We instantiated ABKS, ABE with CP-ABKS, CP-ABE [24] for CP-ABKS, and KP-ABKS, KP-ABE [23] for KP-VABKS, respectively. Finally, we set the example access control policy as "at_1 AND ... AND at_N."

A. Efficiency of ABKS

Asymptotic Complexity of the ABKS Schemes. Table I describes the asymptotic complexity of the ABKS schemes. We observe that in the CP-ABKS scheme, the complexity of KeyGen is almost the same as that of Enc. However, in the KP-ABKS scheme, KeyGen is more expensive than Enc. This discrepancy should serve as a factor when deciding whether to use CP-ABKS or KP-ABKS. In both schemes, the two Search algorithms incur almost the same cost.

ABKS Performance. To evaluate the performance of the ABKS schemes, we ran the experiments on a client machine with Linux OS, 2.93GHz Intel Core Duo CPU (E7500), and 2GB RAM. We varied N, the number of attributes that are involved in the example access control policy, from 1 to 100. We ran each experiment for 10 times so as to average the execution time. Table II shows the execution time of the two ABKS schemes.

From Table II we observe that for both schemes, the keyword encryption algorithm Enc (run by the data owner) is more expensive than that of the keyword search algorithm Search (run by the cloud) with the same N. However, we note that the former algorithm is executed only once for each keyword, whereas the latter algorithm will be performed as many times as needed. This means that even though the keyword search algorithm is relatively light-weight, it can be a performance bottleneck when executed many times. This confirms that the data users should outsource the keyword search operations to the cloud (i.e., taking advantage of the cloud's computational resources).

B. Efficiency of VABKS with Real Data

To demonstrate the feasibility of VABKS in practice, we evaluated it using some real data. The algorithms run by the data owner and the data users (i.e. BuildIndex, TokenGen and Verify) were executed on a client machine with Linux OS, 2.93GHz Intel Core Duo CPU (E7500), and 2GB RAM. The algorithm run by the cloud (i.e., SearchIndex) was executed on a server machine (a laptop) with Windows 7, Intel i5 2.60GHz CPU, and 8GB RAM.

We vary the access control policy ranging from 1 to 50 attributes with step-length 10. In each round, we encrypted all keywords with the identical access control policy. The data consists of 2,019 distinct keywords extracted from 670 PDF documents (papers) from the ACM Digital Library with a total size of 778.1MB. We choose $k = 28$ and $m = 10KB$ so that

KP-ABKS		KeyGen	Enc	TokenGen	Search
	Operations	$3N\mathsf{E} + NH_1$	$(S+4)\mathsf{E} + SH_1$	$(2N+2)\mathsf{E}$	$(2S+2)\mathsf{Pair} + S\mathsf{E}_T$
	Output size	$2N\|G\|$	$(S+3)\|G\|$	$(2N+2)\|G\|$	
CP-ABKS		KeyGen	Enc	TokenGen	Search
	Operations	$(2S+2)\mathsf{E} + SH_1$	$(2N+4)\mathsf{E} + NH_1$	$(2S+4)\mathsf{E}$	$(2N+3)\mathsf{Pair} + N\mathsf{E}_T$
	Output size	$(4S+1)\|G\|$	$(2N+2)\|G\|$	$(2S+2)\|G\|$	

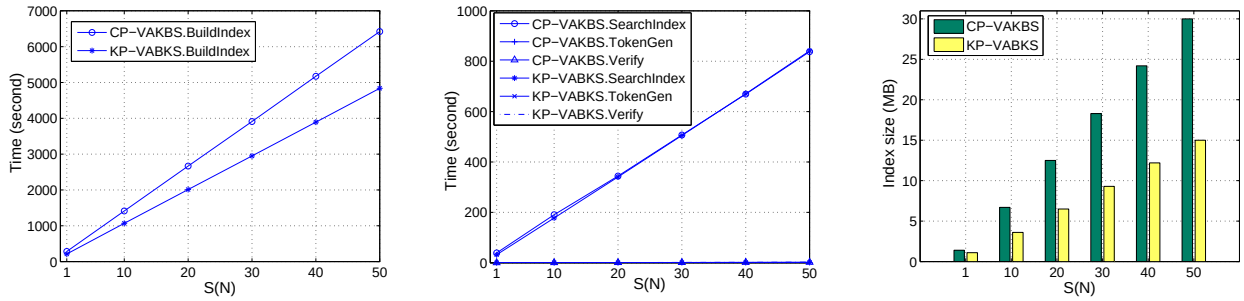
TABLE I

ASYMPTOTIC COMPLEXITIES OF CP-ABKS AND KP-ABKS, WHERE S IS THE NUMBER OF A DATA USER'S ATTRIBUTES AND N IS THE NUMBER OF ATTRIBUTES THAT ARE INVOLVED IN A DATA OWNER'S ACCESS CONTROL POLICY (I.E., THE NUMBER OF LEAVES IN THE ACCESS TREE).

KP-ABKS	KeyGen	0.088	0.786	1.539	2.316	3.081	3.863	4.646	5.396	6.156	6.928	7.705
	Enc	0.108	0.539	1.016	1.492	1.983	2.434	2.939	3.414	3.873	4.358	4.820
	TokenGen	0.073	0.331	0.627	0.917	1.211	1.504	1.778	2.081	2.364	2.673	2.965
	Search	0.049	0.275	0.480	0.711	0.947	1.182	1.425	1.665	1.911	2.153	2.405
	S(N)	1	10	20	30	40	50	60	70	80	90	100
CP-ABKS	KeyGen	0.107	0.686	1.275	1.901	2.525	3.151	3.779	4.405	5.014	5.640	6.262
	Enc	0.121	0.681	1.304	1.923	2.546	3.169	3.787	4.415	5.060	5.673	6.28
	TokenGen	0.088	0.349	0.673	0.932	1.228	1.513	1.823	2.082	2.399	2.665	2.97
	Search	0.061	0.329	0.493	0.728	0.97	1.202	1.441	1.688	1.929	2.182	2.424

TABLE II

EXECUTION TIME (SECOND) OF THE ALGORITHMS IN THE KP-ABKS AND CP-ABKS SCHEMES, WHERE N IS THE NUMBER OF ATTRIBUTES INVOLVED IN THE EXAMPLE ACCESS CONTROL POLICY. THE NUMBER OF DATA USER'S ATTRIBUTES IS ALSO SET TO N, NAMELY $S = N$ IN THE EXPERIMENTS.



(a) Performance of BuildIndex

(b) Performance of TokenGen, SearchIndex and Verify

(c) Size of Index

Fig. 4. Performance of the CP-VABKS and KP-VABKS schemes, where N is the number of attributes involved in the example access control policy. The number of data user's attributes is also set to N, namely $S = N$ in the experiments.

$\frac{m}{n} = \frac{10*8*1024}{2019} \approx 40$ and the false-positive rate is around 4.5×10^{-9}.

Figure 4(a) shows the execution time of BuildIndex that was run by the data owner. We can observe that with the same attribute/policy complexity, CP-VABKS cost more time than that of KP-VABKS when running algorithm BuildIndex. Figure 4(b) plots the execution time of the algorithms run by the data user and the cloud. As algorithm SearchIndex conducts search over keyword ciphertexts one by one sequentially, in the experiments, we generated the search token which matched the 1010th keyword ciphertext. That is, when testing the algorithm SearchIndex, it needs to conduct search operations over 1010 keyword ciphertexts. We can observe that the execution time of TokenGen and Verify is really small, compared with keyword search algorithm SearchIndex. This again confirms that the data user should outsource keyword search operations to the cloud. Figure 4(c) plots the size of index, including 2,019 keyword ciphertexts, bloom filter and signatures. We also see that CP-VABKS consumes more storage space than KP-VABKS while with the same attribute/policy complexity. The reason is that in CP-VABKS scheme, it needs to store extra access trees when serializing CP-ABKS ciphertexts to the storage space.

VI. CONCLUSION

We have introduced a novel cryptographic scheme called verifiable attribute-based keyword search, which is a new tool for secure cloud computing over outsourced encrypted data. This scheme can be used for secure cloud computing: Given the keyword-based data collection, data owners is allowed to control the search capability and use of its outsourced encrypted data according to their access control policy, while authorized data users can outsource the often costly search operations to the cloud and forces the cloud to faithfully execute search operations. Performance evaluation shows that the new tool is practical.

REFERENCES

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE S&P*, pp. 44–, IEEE Computer Society, 2000.

[2] E.-J. Goh, "Secure indexes." Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216/.

[3] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. of ACNS*, pp. 442–455, Springer-Verlag, 2005.

[4] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of*, pp. 79–88, ACM, 2006.

[5] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Proc. of ASIACRYPT*, pp. 577–594, 2010.

[6] K. Kurosawa and Y. Ohtaki, "Uc-secure searchable symmetric encryption," in *Proc. of FC*, pp. 285–298, Springer Berlin / Heidelberg.

[7] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Proc. of FC*, pp. 136–149, 2010.

[8] S. Kamara, C. Papamanthou, and T. Roeder, "Cs2: A searchable cryptographic cloud storage system." Microsoft Technical Report, 2011. http://research.microsoft.com/apps/pubs/?id=148632.

[9] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. of ACM CCS*, pp. 965–976, 2012.

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

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

[12] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proc. of CRYPTO*, pp. 535–552, Springer-Verlag, 2007.

[13] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Proc. of IACM CCSA*, pp. 1249–1259, 2008.

[14] J. Camenisch, M. Kohlweiss, A. Rial, and C. Sheedy, "Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data," in *Proc. of PKC*, pp. 196–214, Springer-Verlag, 2009.

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

[16] E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. of IEEE S&P*, pp. 350–364, 2007.

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

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

[19] F. Bao, R. H. Deng, X. Ding, and Y. Yang, "Private query on encrypted data in multi-user settings," in *Proc. of ISPEC*, pp. 71–85, Springer-Verlag, 2008.

[20] T. Okamoto and K. Takashima, "Hierarchical predicate encryption for inner-products," in *Proc. of ASIACRYPT*, pp. 214–231, Springer-Verlag, 2009.

[21] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Proc. of EUROCRYPT*, pp. 146–162, Springer-Verlag, 2008.

[22] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proc. of EUROCRYPT*, pp. 457–473, 2005.

[23] 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*, pp. 89–98, ACM, 2006.

[24] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. of IEEE S&P*, pp. 321–334, IEEE Computer Society, 2007.

[25] T. Okamoto and K. Takashima, "Fully secure functional encryption with general relations from the decisional linear assumption," in *Proc. of CRYPTO*, pp. 191–208, 2010.

[26] A. B. Lewko and B. Waters, "New proof methods for attribute-based encryption: Achieving full security through selective techniques," in *Proc. of CRYPTO*, pp. 180–198, 2012.

[27] M. Chase, "Multi-authority attribute based encryption," in *Proc. of TCC*, pp. 515–534, Springer-Verlag, 2007.

[28] M. Chase and S. S. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proc. of ACM CCS*, pp. 121–130, ACM, 2009.

[29] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Proc. of CRYPTO*, pp. 111–131, Springer-Verlag, 2011.

[30] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation." Cryptology ePrint Archive, Report 2011/587, 2011. http://eprint.iacr.org/.

[31] D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *Proc. of ACM CCS*, pp. 501–512, ACM, 2012.

[32] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *Proc. of ICC*, pp. 917–922, 2012.

[33] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Proc. of EUROCRYPT*, pp. 440–456, 2005.

[34] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[35] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, July 1970.

[36] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proc. of TCC*, pp. 457–473, Springer-Verlag, 2009.

[37] "The java pairing based cryptography library. http://gas.dia.unisa.it/projects/jpbc/."

APPENDIX A
PROOF OF THEOREM 1

Proof: We show that if there is a polynomial-time adversary $\mathcal{A}$ that wins the SCKA game with advantage μ, then there is a challenger algorithm that solves the DL problem with advantage $\mu/2$. Given a DL instance $(g, h, f, f^{r_1}, g^{r_2}, Q)$, where $g, f, h, Q \leftarrow G$ and $r_1, r_2 \leftarrow \mathbb{Z}_p$, the challenger simulates the SCKA game as follows.

Setup: The challenger sets $g^a = h$ and $g^c = f$ where a and c are unknown, selects $d \leftarrow \mathbb{Z}_p$ and computes $g^b = f^d = g^{cd}$ by implicitly defining $b = cd$. Let H_2 be an one-way hash function and $\mathsf{pm} = (e, g, p, h, f^d, f)$ and $\mathsf{mk} = (d)$.

$\mathcal{A}$ selects an attribute set Atts^* and gives it to the challenger. The random oracle $\mathcal{O}_{H_1}(\mathsf{at}_j)$ is defined as follows:

- If at_j has not been queried before,
  - if $\mathsf{at}_j \in \mathsf{Atts}^*$, select $\beta_j \leftarrow \mathbb{Z}_p$, add $(\mathsf{at}_j, \alpha_j = 0, \beta_j)$ to $\mathcal{O}_{H_1}$, and return g^{β_j};
  - otherwise, select $\alpha_j, \beta_j \leftarrow \mathbb{Z}_p$, add $(\mathsf{at}_j, \alpha_j, \beta_j)$ to $\mathcal{O}_{H_1}$, and return $f^{\alpha_j} g^{\beta_j}$.
- If at_j has been queried before, retrieve (α_j, β_j) from $\mathcal{O}_{H_1}$ and return $f^{\alpha_j} g^{\beta_j}$.

Phase 1: $\mathcal{A}$ can adaptively query the following oracles for polynomially-many times and the challenger keeps a keyword list L_{kw}, which is empty initially.

$\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{T})$: $\mathcal{A}$ gives an access tree T to the challenger. If $F(\mathsf{Atts}^*, \mathsf{T}) = 1$, then the challenger aborts; otherwise, the challenger generates attributes as follows.

Define the following two procedures to determine the polynomial for each node of T:

- PolySat($\mathsf{T}_v, \mathsf{Atts}^*, \lambda_v$): Given secret λ_v, this procedure determines the polynomial for each node of T_v rooted at v when $F(\mathsf{Atts}^*, \mathsf{T}_v) = 1$. It works as follows: Suppose the threshold value of node v is k_v, it sets $q_v(0) = \lambda_v$ and picks $k_v - 1$ coefficients randomly to fix the polynomial q_v. For each child node v' of v, recursively call PolySat($\mathsf{T}_{v'}, \mathsf{Atts}^*, \lambda_{v'}$) where $\lambda_{v'} = q_v(\mathsf{Index}(v'))$.
- PolyUnsat($\mathsf{T}_v, \mathsf{Atts}^*, g^{\lambda_v}$): Given element $g^{\lambda_v} \in G$ where the secret λ_v is unknown, this procedure determines the polynomial for each node of T_v rooted at v when $F(\mathsf{Atts}^*, \mathsf{T}_v) = 0$ as follows. Suppose the threshold value of the node v is k_v. Let V be the empty set. For each child node v' of v, if $F(\mathsf{Atts}, \mathsf{T}_{v'}) = 1$, then set $V = V \bigcup \{v'\}$. Because $F(\mathsf{Atts}, \mathsf{T}_v) = 0$, then

$|V| < k_v$. For each node $v' \in V$, it selects $\lambda_{v'} \leftarrow \mathbb{Z}_p$, and sets $q_v(\mathsf{Index}(v')) = \lambda_{v'}$. Finally it fixes the remaining $k_v - |V|$ points of q_v randomly to define q_v and makes $g^{q_v(0)} = g^{\lambda_v}$. For each child node v' of v,

- if $F(\mathsf{Atts}^*, \mathsf{T}_{v'}) = 1$, then run $\mathrm{PolySat}(\mathsf{T}_{v'}, \mathsf{Atts}^*, q_v(\mathsf{Index}(v'))$, where $q_v(\mathsf{Index}(v'))$ is known to the challenger;
- otherwise, call $\mathrm{PolyUnsat}(\mathsf{T}_{v'}, \mathsf{Atts}^*, g^{\lambda_{v'}})$, where $g^{\lambda_{v'}} = g^{q_v(\mathsf{Index}(v'))}$ is known to the challenger.

With the above two procedures, the challenger runs $\mathrm{PolyUnsat}(\mathsf{T}, \mathsf{Atts}^*, g^a)$, by implicitly defining $q_{\mathsf{root}}(0) = a$. Then for each $v \in \mathsf{lvs}(\mathsf{T})$, the challenger gets $q_v(0)$ if $\mathsf{att}(v) \in \mathsf{Atts}^*$, and gets $g^{q_v(0)}$ otherwise. Because $cq_v(0)$ is the secret share of ac, due to the linear property, the challenger generates credentials for each $v \in \mathsf{lvs}(\mathsf{T})$ as follows:

- If $\mathsf{att}(v) = \mathsf{at}_j$ for some $\mathsf{at}_j \in \mathsf{Atts}^*$: Select $t \leftarrow \mathbb{Z}_p$, set $A_v = f^{q_v(0)} g^{\beta_j t} = g^{cq_v(0)} H_1(\mathsf{att}(v))^t$ and $B_v = g^t$;
- If $\mathsf{att}(v) \notin \mathsf{Atts}^*$ (assuming $\mathsf{att}(v) = \mathsf{at}_j$): Select $t' \leftarrow \mathbb{Z}_p$, set $A_v = (g^{q_v(0)})^{\frac{-\beta_j}{\alpha_j}} (f^{\alpha_j} g^{\beta_j})^{t'}$ and $B_v = g^{q_v(0)\frac{-1}{\alpha_j}} g^{t'}$. Note that (A_v, B_v) is a valid credential because

$$\begin{aligned} B_v &= g^{q_v(0)\frac{-1}{\alpha_j}} g^{t'} = g^{t' - \frac{q_v(0)}{\alpha_j}} \\ A_v &= g^{q_v(0)\frac{-\beta_j}{\alpha_j}} (f^{\alpha_j} g^{\beta_j})^{t'} \\ &= f^{q_v(0)} (f^{\alpha_j} g^{\beta_j})^{\frac{-q_v(0)}{\alpha_j}} (f^{\alpha_j} g^{\beta_j})^{t'} \\ &= f^{q_v(0)} (f^{\alpha_j} g^{\beta_j})^{t' - \frac{q_v(0)}{\alpha_j}} \\ &= g^{cq_v(0)} H_1(\mathsf{att}(v))^{t' - \frac{q_v(0)}{\alpha_j}} \end{aligned}$$

  by implicitly letting $t = t' - \frac{q_v(0)}{\alpha_j}$. Note also that $\mathcal{A}$ cannot construct A_v and B_v without knowing α_j, β_j.

Eventually, the challenger returns $\mathsf{sk} = \{(A_v, B_v) | v \in \mathsf{lvs}(\mathsf{T})\}$ to $\mathcal{A}$.

$\mathcal{O}_{\mathsf{TokenGen}}(\mathsf{T}, w)$: The challenger runs $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{T})$ to get $\mathsf{sk} = (\mathsf{T}, \{A_v, B_v | v \in \mathsf{lvs}(\mathsf{T})\})$, computes $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$, and returns tk to $\mathcal{A}$. If $F(\mathsf{Atts}, \mathsf{T}) = 1$, the challenger adds w to the keyword List L_{kw}.

Challenge phase: $\mathcal{A}$ chooses two keywords w_0 and w_1 of equal length, such that $w_0, w_1 \notin L_{kw}$. The challenger outputs cph^* as:

- Select $\lambda \leftarrow \{0, 1\}$.
- For each $\mathsf{at}_j \in \mathsf{Atts}^*$, set $W_j = (g^{r_2})^{\beta_j}$.
- Set $W' = f^{r_1}$, $W = Q(f^{r_1})^{dH_2(w_\lambda)}$, and $W_0 = g^{r_2}$.
- Set $\mathsf{cph}^* = (\mathsf{Atts}^*, W', W, W_0, \{W_j | \mathsf{at}_j \in \mathsf{Atts}^*\})$ and return cph^* to $\mathcal{A}$.

We note that if $Q = h^{r_1+r_2}$, then cph^* is indeed a legitimate ciphertext for keyword w_λ. The reason is that $W' = f^{r_1} = g^{cr_1}$, $W = Qf^{r_1 dH_2(w_\lambda)} = Qg^{r_1 cdH_2(w_\lambda)} = g^{a(r_1+r_2)} g^{br_1 H_2(w_\lambda)}$, $W_0 = g^{r_2}$, and for $\mathsf{at}_j \in \mathsf{Atts}^*$, $W_j = (g^{r_2})^{\beta_j} = H_1(\mathsf{at}_j)^{r_2}$.

Phase 2: $\mathcal{A}$ continues to query the oracles as in Phase 1. The only restriction is that (T, w_0) and (T, w_1) cannot be the input to $\mathcal{O}_{\mathsf{TokenGen}}$ if $F(\mathsf{Atts}^*, \mathsf{T}) = 1$.

Guess: Finally, $\mathcal{A}$ outputs a bit λ' and gives it to the challenger. If $\lambda' = \lambda$, then the challenger outputs $Q = h^{r_1+r_2}$; otherwise, it outputs $Q \neq h^{r_1+r_2}$.

This completes the simulation. In the challenge phase, if $Q = h^{r_1+r_2}$, then cph^* is a valid ciphertext of w_λ, so the probability of $\mathcal{A}$ outputting $\lambda = \lambda'$ is $\frac{1}{2} + \mu$. If Q is an element randomly selected from G, then cph^* is not a valid ciphertext of w_λ. The probability of $\mathcal{A}$ outputting $\lambda = \lambda'$ is $\frac{1}{2}$ since W is an random element in G. Therefore, the probability of the challenger correctly guessing $Q \stackrel{?}{=} h^{r_1+r_2}$ with the DL instance $(g, h, f, f^{r_1}, g^{r_2}, Q)$ is $\frac{1}{2}(\frac{1}{2} + \mu + \frac{1}{2}) = \frac{1}{2} + \frac{\mu}{2}$. That is, the challenger solves the DL problem with advantage $\mu/2$ if $\mathcal{A}$ wins the SCKA game with an advantage μ. ∎

APPENDIX B
PROOF OF THEOREM 2

Proof: We construct a challenger that exploits the keyword secrecy game as follows:

Setup: The challenger selects $a, b, c \leftarrow \mathbb{Z}_p$, $f \leftarrow G$. Let H_2 be an one-way hash function and $\mathsf{pm} = (e, g, g^a, g^b, g^c, f)$ and $\mathsf{mk} = (a, b, c)$.

The random oracle $\mathcal{O}_{H_1}(\mathsf{at}_j)$ is simulated as follows: If at_j has not been queried before, the challenger selects $\alpha_j \leftarrow \mathbb{Z}_p$, adds $(\mathsf{at}_j, \alpha_j)$ to $\mathcal{O}_{H_1}$, and returns g^{α_j}; otherwise, the challenger retrieves α_j from $\mathcal{O}_{H_1}$ and returns g^{α_j}.

Phase 1: $\mathcal{A}$ can adaptively query the following oracles for polynomially-many times.

$\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{T})$: The challenger generates $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{T}, \mathsf{mk})$ and returns sk to $\mathcal{A}$. It adds T to the list L_{KeyGen}, which is initially empty.

$\mathcal{O}_{\mathsf{TokenGen}}(\mathsf{T}, w)$: The challenger runs $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{T})$ to obtain $\mathsf{sk} = (\mathsf{T}, \{A_v, B_v | v \in \mathsf{lvs}(\mathsf{T})\})$, computes $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$, and returns tk to $\mathcal{A}$.

Challenge Phase: $\mathcal{A}$ selects an attribute set Atts^*. The challenger chooses an access control policy that is represented as T^* such that $F(\mathsf{Atts}^*, \mathsf{T}^*) = 1$, computes $\mathsf{sk}^* \leftarrow \mathsf{KeyGen}(\mathsf{mk}, \mathsf{T}^*)$. By taking as input Atts^* and sk^*, it selects w^* from keyword space uniformly at random, and computes cph^* and tk^* with Enc and TokenGen. Atts^* should satisfy the requirement defined in the keyword secrecy game.

Guess: Finally, $\mathcal{A}$ outputs a keyword w' and gives it to the challenger. The challenger computes $\mathsf{cph}' \leftarrow \mathsf{Enc}(\mathsf{Atts}, w')$ and if $\mathsf{Search}(\mathsf{tk}^*, \mathsf{cph}') = 1$, then $\mathcal{A}$ wins the game.

This finishes the simulation. Suppose $\mathcal{A}$ has already attempted q distinct keywords before outputting w', we can see that the probability of $\mathcal{A}$ winning the keyword secrecy game is at most $\frac{1}{|\mathcal{M}|-q} + \epsilon$. This is because the size of the remaining keyword space is $|\mathcal{M}| - q$, and as the H_2 is an one way secure hash function, meaning deriving w^* from $H_2(w^*)$ is at most a negligible probability ϵ. Therefore, given q distinct keywords $\mathcal{A}$ has attempted, the probability of $\mathcal{A}$ winning the keyword secrecy game is at most $\frac{1}{|\mathcal{M}|-q} + \epsilon$. Thus, our scheme achieves keyword secrecy as in Definition **??**. ∎

APPENDIX C
PROOF OF THEOREM 3 ON CP-ABKS

Proof: We show that the CP-ABE scheme is selectively secure against chosen-keyword attack in the generic bilinear group model, where H_1 is modeled as a random oracle and H_2 is a one-way hash function.

In the SCKA game, $\mathcal{A}$ attempts to distinguish $g^{a(r_1+r_2)g^{br_1H_2(w_0)}}$ from $g^{a(r_1+r_2)g^{br_1H_2(w_1)}}$. Given $\theta \leftarrow \mathbb{Z}_p$, the probability of distinguishing $g^{a(r_1+r_2)}g^{br_1H_2(w_0)}$ from g^{θ} is equal to that of distinguishing g^{θ} from $g^{a(r_1+r_2)}g^{br_1H_2(w_1)}$. Therefore, if $\mathcal{A}$ has advantage ϵ in breaking the SCKA game, then it has advantage $\epsilon/2$ in distinguishing $g^{a(r_1+r_2)}g^{br_1H_2(w_0)}$ from g^{θ}. Thus, let us consider a modified game where $\mathcal{A}$ can distinguish $g^{a(r_1+r_2)}$ from g^{θ}. The modified SCKA game is described as follows:

Setup: The challenger chooses $a, b, c \leftarrow \mathbb{Z}_p$ and sends public parameters (e, g, p, g^a, g^b, g^c) to $\mathcal{A}$. $\mathcal{A}$ chooses an access tree T^*, which is sent to the challenger.

$H_1(\mathsf{at}_j)$ is simulated as follows: If at_j has not been queried before, the challenger chooses $\alpha_j \leftarrow \mathbb{Z}_p$, adds $(\mathsf{at}_j, \alpha_j)$ to O_{H_1} and returns g^{α_j}; otherwise the challenger returns g^{α_j} by retrieving α_j from O_{H_1}.

Phase 1: $\mathcal{A}$ can query $\mathcal{O}_{\mathsf{KeyGen}}$ and $\mathcal{O}_{\mathsf{TokenGen}}$ as follows:
$\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{Atts})$: The challenger selects $r^{(t)} \leftarrow \mathbb{Z}_p$ and computes $A = g^{(ac+r^{(t)})/b}$. For each attribute $\mathsf{at}_j \in \mathsf{Atts}$, the challenger chooses $r_j^{(t)} \leftarrow \mathbb{Z}_p$, computes $A_j = g^{r^{(t)}}g^{\alpha_j r_j^{(t)}}$ and $B_j = g^{r_j^{(t)}}$, and returns $(\mathsf{Atts}, A, \{(A_j, B_j)|\mathsf{at}_j \in \mathsf{Atts}\})$.
$\mathcal{O}_{\mathsf{TokenGen}}(\mathsf{Atts}, w)$: The challenger queries $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{Atts})$ to get $\mathsf{sk} = (\mathsf{Atts}, A, \{(A_j, B_j)|\mathsf{at}_j \in \mathsf{Atts}\})$ and returns $\mathsf{tk} = (\mathsf{Atts}, \mathsf{tok}_1, \mathsf{tok}_2, \mathsf{tok}_3, \{(A'_j, B'_j)|\mathsf{at}_j \in \mathsf{Atts}\})$ where $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s, \mathsf{tok}_2 = g^{cs}$, $\mathsf{tok}_3 = A^s, A'_j = A_j^s$ and $B'_j = B_j^s$ by selecting $s \leftarrow \mathbb{Z}_p$. If $F(\mathsf{Atts}, \mathsf{T}^*) = 1$, the challenger adds w to the keyword List L_{kw}.

Challenge phase: Given two keywords w_0, w_1 of equal length where $w_0, w_1 \notin L_{kw}$, the challenger chooses $r_1, r_2 \leftarrow \mathbb{Z}_p$, and computes secret shares of r_2 for each leaves in T^*. The challenger selects $\lambda \leftarrow \{0, 1\}$. If $\lambda = 0$, it outputs

$$W = g^{cr_1}, W_0 = g^{\theta}, W' = g^{br_2},$$
$$\{(W_v = g^{q_v(0)}, D_v = g^{\alpha_j q_v(0)})|v \in \mathsf{lvs}(\mathsf{T}^*), \mathsf{att}(v) = \mathsf{at}_j\}$$

by selecting $\theta \in \mathbb{Z}_p$; otherwise it outputs

$$W = g^{cr_1}, W_0 = g^{a(r_1+r_2)}, W' = g^{br_2},$$
$$\{(W_v = g^{q_v(0)}, D_v = g^{\alpha_j q_v(0)})|v \in \mathsf{lvs}(\mathsf{T}^*), \mathsf{att}(v) = \mathsf{at}_j\}.$$

Phase 2: This is the same as in the SCKA game.

We can see that if $\mathcal{A}$ can construct $e(g, g)^{\delta a(r_1+r_2)}$ for some g^{δ} that can be composed from the oracle outputs he has already queried, then $\mathcal{A}$ can use it to distinguish g^{θ} from $g^{a(r_1+r_2)}$. Therefore, we need to show that $\mathcal{A}$ can construct $e(g, g)^{\delta a(r_1+r_2)}$ for some g^{δ} with a negligible probability. That is, $\mathcal{A}$ cannot gain non-negligible advantage in the SCKA game.

In the generic group model, ψ_0 and ψ_1 are random injective maps from $\mathbb{Z}_p$ into a set of p^3 elements. Then the probability of $\mathcal{A}$ guessing an element in the image of ψ_0 and ψ_1 is negligible.

a	$r_j^{(t)}$	$s(ac + r^{(t)})/b$	cr_1
b	$r^{(t)} + \alpha_j r_j^{(t)}$	$s(r_j^{(t)})$	$q_v(0)$
c	$(ac + r^{(t)})/b$	$s(r^{(t)} + \alpha_j r_j^{(t)})$	$\alpha_j q_v(0)$
α_j	cs	$s(a + bH_2(w))$	br_2

TABLE III
POSSIBLE TERMS FOR QUERYING GROUP ORACLE G_T

Recall that $G = \{\psi_0(x)|x \in \mathbb{Z}_p\}$ and $G_T = \{\psi_1(x)|x \in \mathbb{Z}_p\}$. Hence, let us consider the probability of $\mathcal{A}$ constructing $e(g, g)^{\delta a(r_1+r_2)}$ for some $\delta \in \mathbb{Z}_p$ from the oracle outputs he has queried.

We list all terms that can be queried to the group oracle G_T in Table III. Let us consider how to construct $e(g, g)^{\delta a(r_1+r_2)}$ for some δ. Because r_1 only appears in the term cr_1, δ should contain c in order to construct $e(g, g)^{\delta a(r_1+r_2)}$. That is, let $\delta = \delta' c$ for some δ' and $\mathcal{A}$ wishes to construct $e(g, g)^{\delta' ac(r_1+r_2)}$. Therefore, $\mathcal{A}$ needs to construct $\delta' acr_2$, which will use terms br_2 and $(ac+r^{(t)})/b$. Because $(br_2)(ac+r^{(t)})/b = acr_2 + r^{(t)}r_2$, $\mathcal{A}$ needs to cancel $r^{(t)}r_2$, which needs to use the terms $\alpha_j, r^{(t)} + \alpha_j r_j^{(t)}$, $q_v(0)$ and $\alpha_j q_v(0)$ because $q_v(0)$ is the secret share of r_2 according to T^*. However, it is impossible to construct $r^{(t)}r_2$ with these terms because $r^{(t)}r_2$ only can be reconstructed if the attributes corresponding to $r_j^{(t)}$ of $r^{(t)} + \alpha_j r_j^{(t)}$ satisfies the access tree T^*.

Therefore, we can conclude that $\mathcal{A}$ gains a negligible advantage in the modified game, which means that $\mathcal{A}$ gains a negligible advantage in the SCKA game. This completes the proof. ∎

APPENDIX D
PROOFS OF THEOREM 5, THEOREM 6, THEOREM 7 AND THEOREM 8 ON VABKS

A. Proof of Theorem 5 on VABKS

Proof: We show that if there exists a polynomial-time algorithm $\mathcal{A}$ breaks VABKS's data secrecy with the advantage ρ, then we can break either CPA security for ABE or CPA security for SE with the advantage $\frac{\rho}{n^2}$ where n is the number of data files to be encrypted.

The challenger proceeds the conventional CPA security game with $\mathcal{A}$. In the challenge phase, suppose $\mathcal{A}$ presents two data collections $\mathsf{D}_0 = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}_0 = \{\mathsf{F}_{01}, \ldots, \mathsf{F}_{0n}\})$, $\mathsf{D}_1 = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}_1 = \{\mathsf{F}_{11}, \ldots, \mathsf{F}_{1n}\})$, $\{I_{\mathsf{Enc}}\}_l$ and $\{I'_{\mathsf{Enc}}\}_n$. The challenge selects $\lambda \leftarrow \{0, 1\}$ and encrypts FS_λ with the ABE and $\{I'_{\mathsf{Enc}}\}_n$.

Now let us consider the advantage of $\mathcal{A}$ correctly guessing λ. As we know, given two messages, the advantage of distinguishing which message was encrypted by the hybrid encryption of ABE and SE is equal. Therefore, given two sets of data files FS_0 and FS_1, if the advantage of distinguishing which data set was encrypted is ρ, then the advantage of distinguishing which data file was encrypted is $\frac{\rho}{n^2}$ by selecting one data file from FS_0 and one from FS_1.

Therefore, we can see that if $\mathcal{A}$ breaks VABKS's data secrecy of with a non-negligible advantage ρ, then the advantage

of breaking CPA security for ABE or CPA security for SE is $\frac{\rho}{n^2}$ –a non-negligible probability, which contracts the assumption that ABE is CPA-secure and SE is CPA-secure. ■

B. Proof of Theorem 6 on VABKS

Proof: We show that if there exists a polynomial-time algorithm $\mathcal{A}$ breaks the selective security against chosen-keyword attack of ABKS with the advantage ρ, then we can break the selective security against chosen-keyword attack game of ABKS with the advantage of $\frac{\rho}{l^2}$, given that ABE is CPA-secure and H is a secure pseudorandom generator.

The challenger proceeds selective security against chosen-keyword attack game with $\mathcal{A}$. In the challenge phase, suppose $\mathcal{A}$ presents two data collections $\mathsf{D}_0 = (\mathsf{KS}_0 = \{\mathsf{KS}_{01}, \ldots, \mathsf{KS}_{0l}\}, \mathsf{MP}, \mathsf{FS})$, and $\{I'_{\mathsf{Enc}}\}_n$. The challenge selects $\lambda \leftarrow \{0,1\}$ and encrypts KS with ABKS, and generates $\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}$ and σ_i for each keyword group.

Since ABE is CPA-secure and H is a secure pseudorandom generator, the probability of $\mathcal{A}$ inferring λ via $\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}$ is negligible. Then let us consider the advantage of $\mathcal{A}$ correctly guessing λ from keyword ciphertexts. As we know, given two keywords, the advantage of distinguishing which keyword was encrypted by ABKS is equal. Therefore, given two keyword sets KS_0 and KS_1, if the advantage of distinguishing which keyword set was encrypted is ρ, then the advantage of distinguishing which keyword was encrypted is $\frac{\rho}{l^2}$ by selecting one keyword from KS_0 and one from KS_1.

Therefore, we can see that if $\mathcal{A}$ breaks VABKS's selective security against chosen-keyword attack with a non-negligible advantage ρ, then the advantage of breaking ABKS's selective security against chosen-keyword attack is $\frac{\rho}{l^2}$ –a non-negligible probability, which contracts the assumption that ABKS achieve selective security against chosen-keyword attack, given that ABE is CPA-secure and H is a secure pseudorandom generator. ■

C. Proof of Theorem 7 on VABKS

Proof: We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking VABKS'skeyword secrecy, then it breaks the assumption that ABKS achieves keyword secrecy.

Suppose $\mathcal{A}$ presents a data collection $\mathsf{D} = (\mathsf{KS} = \{\mathsf{KS}_1, \ldots, \mathsf{KS}_l\}, \mathsf{MP}, \mathsf{FS})$, $\{I_{\mathsf{Enc}}\}_l$ and $\{I'_{\mathsf{Enc}}\}_n$. The challenger simulates the keyword secrecy game as in B, where the keyword space consists of keywords specified by FS. We can see that the probability of $\mathcal{A}$ inferring the keyword from a search token and corresponding keyword ciphertext is equal to that of ABKS. Therefore, if in VABKS$\mathcal{A}$ guesses the keyword from the search token and corresponding keyword ciphertext with the probability more than $\frac{1}{|\mathcal{M}|-q} + \epsilon$ after guessing q distinct keywords, then the probability of guessing the keyword from the search token and keyword ciphertext in ABKS is more than $\frac{1}{|\mathcal{M}|-q} + \epsilon$ after guessing q distinct keywords, which contracts the assumption that ABKS achieves keyword secrecy. ■

D. Proof of Theorem 8 on VABKS

Proof: We show that under the assumptions that Sig is unforgeable, any polynomial-time adversary $\mathcal{A}$ presents an incorrect search result and succeeds in the verification with negligible probability.

The challenger proceeds the verifiability game, where $\mathcal{A}$ provides the keyword-based data $\mathsf{D} = (\mathsf{KS} = \{\mathsf{KS}_1, \ldots, \mathsf{KS}_l\}, \mathsf{MP} = \{\mathsf{MP}(w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}, \mathsf{FS} = \{\mathsf{F}_1, \ldots, \mathsf{F}_n\})$, $\{I_{\mathsf{Enc}}\}_l$ and $\{I'_{\mathsf{Enc}}\}_n$. The challenger runs $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}) \leftarrow \mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$, and gives $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}})$ to $\mathcal{A}$.

In the challenge phase, with w^* and I^*_{Enc} from $\mathcal{A}$, the challenger selects I^*_{KeyGen} such that $F(I^*_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$ where I^*_{Enc} is selected by $\mathcal{A}$, generates credential sk^* with I^*_{KeyGen} and returns to $\mathcal{A}$ a search token tk^* by running $\mathsf{tk}^* \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w^*)$. $\mathcal{A}$ returns $(\mathsf{rslt}^*, \mathsf{proof}^*)$ to the challenger.

Suppose that $(\mathsf{rslt}^*, \mathsf{proof}^*)$ succeeds in the verification. That is, $1 \leftarrow \mathsf{Verify}(\mathsf{sk}^*, w^*, \mathsf{tk}^*, \mathsf{rslt}^*, \mathsf{proof}^*)$. Let us consider the probability of $\mathcal{A}$ cheating with incorrect search result.

First, we claim that the global signature σ and random keyword ciphertexts $\mathsf{cph}_{\mathsf{BF}_1}, \ldots, \mathsf{cph}_{\mathsf{BF}_l}$ are included in proof^* without being manipulated; otherwise we can break the unforgeability of Sig.

Second, let us consider the search result within each group with respect to access control policies, i.e. $i = 1, \ldots, l$:

- If there exists no keyword ciphertext matched the search token tk^*, then we claim that $\mathcal{A}$ cannot cheat the challenger with some keyword ciphertext and data ciphertexts in order to make VABKS.Verify output 1. The reason is that $\mathcal{A}$ cannot forge a keyword signature σ_w for the keyword ciphertext and data ciphertexts; otherwise, we can break the unforgeability of Sig.
- If there exists a keyword ciphertext matched the search token tk^*, then we claim that $\mathcal{A}$ cannot cheat the challenger with a null search result in order to make VABKS.Verify output 1. Suppose $\mathcal{A}$ returns a null result and the proof $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i})$. Since BF'_i cannot be manipulated due to σ_{BF_i}, the unmasked bloom filter indicates that w^* is a member within the group. The challenger downloads $\mathsf{cph}_{w_1}, \ldots, \mathsf{cph}_{w_{|\mathsf{KS}_i|}}$ and σ_i without being manipulated; otherwise we break the Sig's unforgeability. Then the challenger can conduct the search operation with each keyword ciphertext, and VABKS.Verify will output 0. That is, if there exists keyword ciphertext matched the search token, $\mathcal{A}$ returns a null result, then it cannot make VABKS.Verify output 1.

To sum up, in order to make VABKS.Verify output 1, $\mathcal{A}$ has to faithfully execute search operations and return the search result honestly; otherwise, we will break Sig's unforgeability. ■