VABKS: 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. Since the cloud is not fully trusted, the outsourced data should be encrypted, which however brings a range of problems, such as: How can the 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 on their behalf? Towards ultimately addressing these problems, in this paper we propose a novel cryptographic solution, called *verifiable attribute-based keyword search* (VABKS). This solution allows a data user, whose 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 user's the search operations. We define VABKS's security properties and introduce concrete constructions that are proven to satisfy them. Performance evaluation shows that the proposed schemes are practical and deployable.**

I. Introduction

Cloud computing allows data owners to use massive data storage and vast computation capabilities at a very convenient price. Despite its benefits, data outsourcing deprives data owners of direct control over their outsourced data. To alleviate such concerns, data owners should adopt encryption and encrypt their data before storing it into the cloud. However, encryption alone may severely hinder several functionalities cloud solutions have accustomed users to. For instance, it would be impossible to search over data owner's outsourced encrypted data. In addition, a data owner may not be able to grant search capabilities to data users according to a specified access control policy (e.g., stating that only certain users can search on sensitive data). While the cloud may search on encrypted data thanks to current cryptographic techniques, it is not clear how to verify whether it faithfully followed the requested search operations. As we will elaborate later, existing solutions cannot achieve these goals simultaneously.

A. Our Contributions

We propose a novel cryptographic solution by which data owners can 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 users' search operations. More specifically, the solution allows a data user with proper credentials (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 or not the cloud has faithfully executed the user's search operations. The solution is centered on a novel cryptographic scheme, *verifiable attribute-based keyword search* (VABKS).

In addition to formally define security properties of VABKS, we present the VABKS construction 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. Our application of bloom filter and digital signature allows data users to *efficiently* verify that the cloud honestly conducted the users' search operations over some data owner's outsourced encrypted data. Experimental evaluation shows that the VABKS (and VABKS) schemes are practical.

B. Related Work

To the best of our knowledge, no solution is adequate for what we want to achieve. Nevertheless, we briefly review the relevant techniques below.

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]–[10]) and searchable encryption in the public-key setting (e.g., [11]–[15]). In these solutions, the data owner generates some tokens that can be used by a data user to search over the data owner's encrypted data. Several variants (e.g., [16]–[19]) have been proposed to support complex queries. Moreover [4], [20] considered searchable encryption in the multi-users setting, where the data owner can enforce an access control policy by distributing (stateful) secret keys to authorized users directly and revoke the secret keys if necessary. However, all these solutions do not solve the problem studied in the present paper because (1) some solutions require interactions between data users and data owners (or trusted proxy, e.g., trapdoor generation entity [15]) to grant search capabilities, and (2) all these silutions (except [10]) assume that the server faithfully executed search operations. In contrast, our solution allows a data user with proper credentials to issue search tokens by which the cloud can perform keyword search on behalf of the user, *without* requiring any interaction with the data owner. Moreover, the data user can verify whether or not the cloud has faithfully executed the user's search operations. This is true even for

the powerful technique called predicate encryption [21], [22], which does not offer the desired verifiability.

Attribute-Based Encryption (ABE). This technique allows entities with proper credentials to decrypt a ciphertext in question [23]. It has two variants, depending on how access control is enforced, called key-policy ABE (KP-ABE), where the decryption key is associated with an access control policy [24], and ciphertext-policy ABE (CP-ABE), where the ciphertext is associated with an access control policy [25]. ABE is a popular method for enforcing access control policies via cryptographic means, and has been enriched with various features (e.g., [26]–[29]). In this paper, we extend ABE with the novel feature of keyword search, namely *attribute-based keyword search*, by which keywords are encrypted with some access control policies so that only data users with proper cryptographic credentials can generate tokens that can be used to searcher over the outsourced encrypted data. This effectively prevents data owners from knowing the keywords a data user is searching for while requiring no interaction between data users and data owners/trusted authorities. This is even in contrast to [15], where data users should interact with data owners/trusted authorities to obtain search tokens obliviously.

Verifiable Keyword Search. Recently, verifiable keyword search solutions have been proposed in [30]–[32], where each keyword is represented as a root of a polynomial. It is possible to check whether a keyword is present or not by evaluating the polynomial on the keyword and verifying if its output is zero. However, these approaches work only when keywords are sent in the cleartext to the cloud, and are not suitable for our purpose because the cloud should not know the keywords involved. It is worth mentioning that the secure verifiable keyword search in the symmetric-key setting [10] can be *insecure* in the public-key setting because the attacker can infer keywords in question via an *off-line* keyword guessing attack (in lieu of the off-line dictionary attack against passwords).

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 the VABKS construction and analyzes its security. Section V evaluates the performance of the ABKS and VABKS schemes. Section VI concludes the paper.

II. Preliminaries

Let $a \leftarrow S$ denote selecting an element a from a set S uniformly at random, $||$ denote the concatenation operation and $\mathsf{string}(S)$ denote the concatenation of elements of S ordered by their hash values. Let $\mathsf{U} = \{\mathsf{at}_1, \ldots, \mathsf{at}_n\}$ be the set of attributes, based on which access control policies are specified.

A. Cryptographic Assumptions and Structure

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 $\mathcal{A}$ can determine if $Q \stackrel{?}{=} h^{r_1+r_2}$ 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]. 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 $G_T = \{\psi_1(x) | x \in \mathbb{Z}_p\}$. There is an 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 an ℓ-bit seed and generates a m-bit string that cannot be distinguished from a m-bit random string by any polynormial-time algorithm (in ℓ).

Attribute-based Encryption (ABE). Let ABE be a secure attribute-based encryption scheme [24], [25], such that $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ where Setup is to initialize the system parameter and master key, KeyGen is to generate credentials for users, Enc is to encrypt the data with the access control policy and Dec is to decrypt the ciphertext with the user's credentials.

Symmetric Encryption (SE) [34]. Let SE be a secure symmetric encryption, such that $\mathsf{SE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ where KeyGen is to generate symmetric key, Enc is to encrypt the message and Dec is to decrypt the ciphertext.

Digital Signature (Sig) [34]. Let Sig be a secure digital signature, such that $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, where KeyGen is to generate public/private key, Sign is to generate a signature for the message and Verify is to verify whether the message is matched the signature.

B. Bloom Filter for Membership Query

A Bloom filter [35] is a data structure for succinctly representing a static set of items while allowing membership queries. A m-bit Bloom filter is an array of m bits, that are initially set to 0. It uses k independent universal hash functions $H'_1, \ldots, H'_k$ of range $\{0, \ldots, m-1\}$. For each element $w \in S = \{w_1, \ldots, w_n\}$, the bits corresponding to $H'_j(w)$ are set to 1 for $1 \leq j \leq k$. To check whether w is an element of S, it verifies if all bits corresponding to $H'_j(w)$, $1 \leq j \leq k$, 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). Suppose the k hash functions are perfectly random and n elements are hashed into the m-bit Bloom filter, the false positive rate is $(1 - (1 - \frac{1}{m})^{km})^k \approx (1 - e^{-kn/m})^k$. The optimal number of hash functions minimizing the false positive rate is $k = (\ln 2)m/n$, and the minimal false positive rate 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 1 if w is an element of S, and 0 otherwise.

C. Access Trees for Representing Access Control Policies

Access trees are convenient for representing access control policies [24]. In an access tree, a leaf is associated with an attribute and an inner node represents a threshold gate. Let num_v be the number of children of the node v, and label the children from the left to the right as 1 to num_v. Let k_v be the threshold value associated with node v, so that $1 \leq k_v \leq \mathsf{num}_v$, where $k_v = 1$ represents the OR gate and $k_v = num_v$ represents the AND gate as two special cases. Let $\mathsf{parent}(v)$ denote the parent of 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 $\mathsf{Atts} \subseteq \mathsf{U}$, Let $F(\mathsf{Atts}, \mathsf{T}_v) = 1$ indicate that Atts satisfies the access control policy represented by 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 $\forall\, j \in I, F(\mathsf{Atts}, \mathsf{T}_{v_j}) = 1$, then set $F(\mathsf{Atts}, \mathsf{T}_v) = 1$; otherwise, set $F(\mathsf{Atts}, \mathsf{T}_v) = 0$.

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

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

The algorithm generates 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) as follows.

- 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.
- Else if v is a leaf of T, set $q_v(0) = q_{\mathsf{parent}(v)}(\mathsf{ind}(v))$.
- Otherwise 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 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 executes 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.
- Otherwise, $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_v) = 1$ and it executes below:
  - If v is a leaf, set $E_v = E_{u_j}(0) = e(g,h)^{q_{u_j}(0)}$ where $v = u_j$ for some j.
  - Otherwise, v is an inner node. 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 has an associated value $E_{\mathsf{root}} = e(g,h)^{q_{\mathsf{root}}(0)} = e(g,h)^s$.

III. Attribute-Based Keyword Search

This building-block technique, called attribute-based keyword search (ABKS), allows data owners to specify access control policies with respect to their data and keyword search functions when outsourcing their encrypted data to the cloud, The data users who possess attributes satisfying a data owner's access control policy can search over encrypted keywords without interacting with the data owner.

This building-block technique has two variants: Key-Policy ABKS (KP-ABKS) where the cryptographic credentials are associated to an access control policy, and Ciphertext-Policy ABKS (CP-ABKS) where the ciphertext is associated to an access control policy. To unify the presentation, let I_{Enc} denote the input to encryption function Enc and I_{KeyGen} to denote the input to key generation function KeyGen. In the case of CP-ABKS, I_{Enc} and I_{KeyGen} are respectively the access tree and attribute set. In the case of KP-ABKS, I_{Enc} and I_{KeyGen} are respectively the attribute set and access tree. Let $F(I_{\mathsf{KeyGen}}, I_{\mathsf{Enc}}) = 1$ denote that I_{KeyGen} satisfies I_{Enc} in CP-ABKS and that I_{Enc} satisfies I_{KeyGen} in KP-ABKS.

A. ABKS Definition and Security

Definition 1: An ABKS operates in the following model: A data owner outsources its encrypted keyword to the cloud, a data user generates the search token with respect to some keyword and retrieves encrypted data from the cloud, and a cloud provides data storage and retrieval service upon request. The ABKS consists of algorithm as follows:

- $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Setup}(1^\ell)$: 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 for a user.
- $\mathsf{cph} \leftarrow \mathsf{Enc}(w, I_{\mathsf{Enc}})$: Taking as input keyword w and I_{Enc}, this algorithm outputs the encrypted keyword cph.
- $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$: Taking as input credential sk and keyword w, a legitimate (or authorized) data user uses this algorithm to generates a search token tk.
- $\{0,1\} \leftarrow \mathsf{Search}(\mathsf{cph}, \mathsf{tk})$: The cloud uses this algorithm to determine whether the encrypted keyword cph and the

token tk correspond to the same keyword. Return 1 if so, and 0 otherwise.

Correctness of ABKS 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.

The threat model of ABKS is the following: data owners and authorized data users are trusted, whereas the cloud is semi-trusted (i.e., trusted but curious). ABKS security requires that the cloud learn nothing but the search results. Specifically, given a probabilistic polynomial-time adversary $\mathcal{A}$ modeling the semi-trusted cloud, an ABKS scheme should satisfy the following intuitive security requirements:

- Selective security against chosen-keyword attack: This notion is to capture that the adversary $\mathcal{A}$ cannot deduce any information about keyword only with keyword ciphertexts (i.e. without being any matched search tokens) in the selective security model, other than what it already knows from previous results, e.g. querying search tokens by adaptively choosing keywords. In other words, $\mathcal{A}$ cannot distinguish the encryption of two challenge keywords of its choice. Here the selective security model [36] means that $\mathcal{A}$ must determine I_{Enc} it intends to attack before the system parameters are initiated. We formalize this security property via the following selective chosen-keyword attack game .
- Keyword secrecy: In the public-key setting, it is impossible to protect the information encoded by search tokens (aka. predicate privacy [37]) due to the *keyword guessing attack*: $\mathcal{A}$ can encrypt any keyword of his choice and check whether the resulting keyword ciphertext and the target token have the same keyword. Therefore, we use a weaker notion, *keyword secrecy*, that assures the probability of $\mathcal{A}$ learning the keyword from the ciphertext as well as the search token is no more than that of one 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*

The basic idea underlying the ABKS construction is to separate each keyword ciphertext and search token into two parts: one is associated to the keyword and the other is associated to the attributes (or access control policy). If the data user's attributes satisfy the access control policy, it can determine whether the search token matches the encrypted keyword.

To further highlight the basic idea, let us consider KP-ABKS as example. Assume that $H_1 : \{0,1\}^* \to G$ is a secure hash function modeled as random oracle, and $H_2 : \{0,1\}^* \to \mathbb{Z}_p$ is a secure one-way hash function. A data user's 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 access tree T. The keyword ciphertext and search token are generated as follows:

- Keyword w is encrypted into two parts: one is to "blend" 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 to the attribute set Atts by letting $W_j = H_1(\mathsf{at}_j)^{r_2}$ for each $\mathsf{at}_j \in \mathsf{Atts}$. The two parts are tied to r_2.

- Given a set of credentials, a search token for keyword w is generated with two parts: one is associated to the keyword, $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s$ and $\mathsf{tok}_2 = g^{cs}$ by selecting $s \leftarrow \mathbb{Z}_p$, and the other is associated to the credentials by letting $A'_v = A_v^s, B'_v = B_v^s$ for each $v \in \mathsf{lvs}(\mathsf{T})$. The two parts are tied to s.

As long as the attribute set satisfies the access tree T, the cloud can use A'_v, B'_v and W_0, W_j to recover $e(g,g)^{acr_2 s}$, which can be used to test the keyword equality as elaborated below.

KP-ABKS **Construction.** Let ℓ be the primary security parameter. It consists of the following algorithms.

$\mathsf{Setup}(1^\ell)$: 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 $g \leftarrow G$, and sets

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

$\mathsf{KeyGen}(\mathsf{mk}, \mathsf{T})$: 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 $B_v = g^t$. It sets

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

$\mathsf{Enc}(w, \mathsf{Atts})$: 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 $W_j = H_1(\mathsf{at}_j)^{r_2}$. It sets

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

$\mathsf{TokenGen}(\mathsf{sk}, w)$: 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 $\mathsf{tok}_2 = g^{cs}$. It sets

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

$\mathsf{Search}(\mathsf{tk}, \mathsf{cph})$: Given the attribute set Atts included in cph, this algorithm selects a 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 returns 1 if $e(W', \mathsf{tok}_1)E_{\mathsf{root}} = e(W, \mathsf{tok}_2)$, and 0 otherwise.

Correctness of KP-ABKS can be verified as follows:

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

Security of KP-ABKS is assured by the following theorems, whose proofs are given in 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.** Let ℓ be the primary security parameter. It consists of the following algorithms.

$\mathsf{Setup}(1^\ell)$: This algorithm selects a bilinear group $e : G \times G \to G_T$, 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 $g \leftarrow G$, and sets

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

$\mathsf{KeyGen}(\mathsf{mk}, \mathsf{Atts})$: 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 $B_j = g^{r_j}$. It sets

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

$\mathsf{Enc}(w, \mathsf{T})$: 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 $D_v = H_1(\mathsf{att}(v))^{q_v(0)}$. It sets

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

$\mathsf{TokenGen}(\mathsf{sk}, w)$: 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 $B'_j = B_j^s$. It sets

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

$\mathsf{Search}(\mathsf{tk}, \mathsf{cph})$: Given an attribute set Atts included in tk, this algorithm selects a 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 $e(W_0, \mathsf{tok}_2) = e(W, \mathsf{tok}_1)E_{\mathsf{root}} e(\mathsf{tok}_3, W')$, and 0 otherwise.

The correctness of CP-ABKS can be verified similar to that of KP-ABKS. Security of CP-ABKS is assured by the following theorems. 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 [33].

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

IV. Verifiable Attribute-Based Keyword Search

In the model of ABKS, the cloud is semi-trusted. VABKS achieves the goal of ABKS in the presence of possibly malicious cloud, which may cheat in the search operation.

A. System Model and Threat Model

We consider the system model as illustrated in Figure 1, which involves four entities: the data owner, who outsources its encrypted data which is indexed by keywords that are also encrypted but can be searched by the data users with the proper credentials; the cloud, which provides storage services and can conduct the actual keyword search operations on behalf of the data users; the data user, who is to retrieve the data owner's encrypted data (i.e. keyword ciphertext and the associated data ciphertexts) according to some keyword; the trusted authority that issues credentials to the data owners/users. We assume that the credentials are sent over authenticated private channels (which can be achieved through another layer of mechanisms).

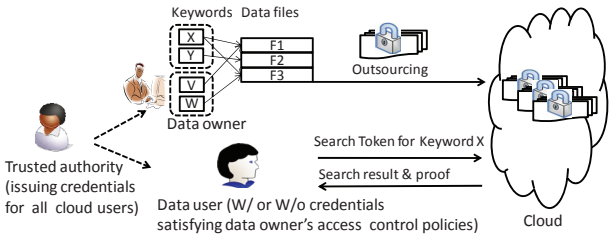


Fig. 1. System model for verifiable attribute-based keyword search over outsourced encrypted data, where keywords X, Y and V, W have different access control policies.

The data owners are naturally trusted. We assume that both authorized and unauthorized data users are semi-trusted, meaning that they may try to derive some sensitive information of interest. We assume that the cloud is *not* trusted as it may manipulate the search operations (already implying that it manipulates the stored data).

B. VABKS *Definition and Security*

Let $\mathsf{D} = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS})$ denote the inverted index and the set of data files. $\mathsf{KS} = \{\mathsf{KS}_1, \ldots, \mathsf{KS}_l\}$ is a set of l keyword sets (also referred as "keyword group"), in which elements are encrypted with the same access control policy. $\mathsf{MP} = \{\mathsf{MP}(w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}$ is the set of $\mathsf{MP}(w), w \in \cup_{i=1}^{l} \mathsf{KS}_i$, and $\mathsf{MP}(w)$ consists of the set of identifiers for identifying data files associated with keyword w. $\mathsf{FS} = \{\mathsf{F}_1, \ldots, \mathsf{F}_n\}$ is the set of n data files.

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

- $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Init}(1^\ell)$: The trusted authority runs this algorithm to initialize the system.
- $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: The trusted authority runs this algorithm issue credentials sk to a data user/owner.
- $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}) \leftarrow \mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$: A data owner runs this algorithm to encrypt $\mathsf{D} = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS})$ and obtain data ciphertexts $\mathsf{D}_{\mathsf{cph}}$, encrypted index Index and auxiliary information Au, where $\{I_{\mathsf{Enc}}\}_l$ is the set of access control policies to encrypt l keyword groups in KS and $\{I'_{\mathsf{Enc}}\}_n$ is the set of access control policies to encrypt n data files in FS.
- $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$: With credential sk, the data user runs this algorithm to issue search token tk for keyword w.
- $(\mathsf{proof}, \mathsf{rslt}) \leftarrow \mathsf{SearchIndex}(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}, \mathsf{tk})$: The cloud uses this algorithm to perform search operations over encrypted index Index on behalf of a data user. It outputs the search result rslt and a proof proof.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{sk}, w, \mathsf{tk}, \mathsf{rslt}, \mathsf{proof})$: The data user runs this algorithm to verify that $(\mathsf{rslt}, \mathsf{proof})$ is valid with respect to search token tk.

Correctness of VABKS 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, security of VABKS is specified by the following four requirements against untrusted cloud $\mathcal{A}$.

- Data secrecy: Even given encrypted keywords and search tokens, $\mathcal{A}$ still cannot learn any information (in a computational sense) about encrypted data files. This definition can be formalized similar to chosen-plaintext security game where 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 $|\mathsf{FS}_0| = |\mathsf{FS}_1|$.
- Selective security against chosen-keyword attack: Without seeing the corresponding search tokens, $\mathcal{A}$ cannot guess the keyword that is encrypted. This requirement is extended from selective security against chosen-keyword attack of ABKS.
- Keyword secrecy: Even given the encrypted data files, the probability of $\mathcal{A}$ learning the keyword from a keyword ciphertext and the search tokens is no more than that of a random guess. This security definition also can be extended from the keyword secrecy of ABKS.
- Verifiability: If $\mathcal{A}$ returns an incorrect search result, then the cheating behavior can be detected with an 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.

- $\mathcal{O}_{\mathsf{KeyGen}}(I_{\mathsf{KeyGen}})$: 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

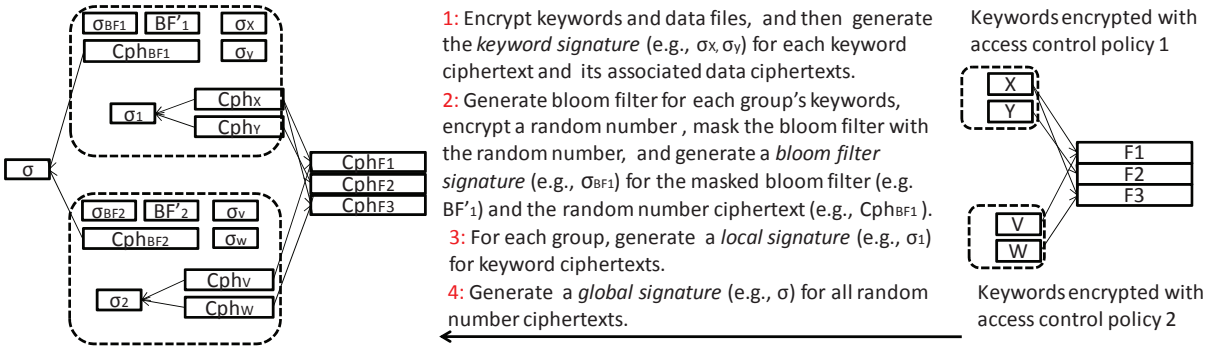


Fig. 2. Basic idea for achieving verifiability, where data files F_1, F_2, F_3 were encrypted to $\mathsf{cph}_{\mathsf{F}_1}, \mathsf{cph}_{\mathsf{F}_2}, \mathsf{cph}_{\mathsf{F}_3}$, keywords X, Y were encrypted to $\mathsf{cph}_X, \mathsf{cph}_Y$ with access control policy 1, and keywords V, W were encrypted to $\mathsf{cph}_V, \mathsf{cph}_W$ with access control policy 2. Given a search token tk, for each group i, the cloud provides $(\sigma_w, \mathsf{cph}_{\mathsf{BF}_i})$ as the proof when it finds some keyword ciphertext cph_w that matches tk, and $(\mathsf{cph}_{\mathsf{BF}_i}, \mathsf{BF}'_i, \sigma_{\mathsf{BF}_i})$ otherwise.

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

A trivial solution is that data users download all keyword ciphertexts and check whether any of keyword ciphertexts and the search token has the same keyword, which however incurs prohibited communication and computation overhead. Instead, we let a data user verify that (1) the cloud performed search operations over all keyword ciphertexts, and (2) the cloud honestly returned the search result for each group, which could be either null or one keyword ciphertext and its associated data ciphertexts. The basic idea is further illustrated in Figure 2. More specifically, the data owner uses 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. This prevents the cloud from returning incorrect data ciphertexts in question as the search result.
- For each group, one bloom filter is built from its keywords. This allows data users to *efficiently* check that the queried keyword was indeed not in the keyword group when the cloud returns a null search result, *without* downloading all keyword ciphertexts from the cloud. A random number is selected and encrypted with the same access control policy as keywords. The random number masks 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 allows a data user to verify integrity of all random number ciphertexts.
- The fourth type of signature, called *local signature*, is generated for all keyword ciphertexts within the same group. This signature is to validate integrity of keyword ciphertexts within the group.

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

In algorithm Verify of Figure 3, note that when an authorized data user verifies a null search result for the group $\{\mathsf{cph}_w | w \in \mathsf{KS}_i\}$ with respect to a search token that corresponds to 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 due to the false-positive of the Bloom filter. To validate the search result in this case, algorithm Verify has to download $\{\mathsf{cph}_w | w \in \mathsf{KS}_i\}$, and checks one by one. We stress that this does not incur any significant unnecessary communications from the perspective of amortization because we can set the false-positive rate as low as possible by choosing appropriate m and k (i.e., the "wasted" bandwidth communication and computation cost are proportional to this false-positive rate). For example, in our experiment we set the false-positive rate to be 4.5×10^{-9}.

D. Security Analysis

We show that the VABKS scheme satisfies the security requirements with the following theorems, whose proofs 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 and SE are secure against chosen-plaintext attack, 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

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

$\underline{\mathsf{KeyGen}}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: The attribute authority runs $\mathsf{ABE.sk} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mk}, I_{\mathsf{KeyGen}})$ and $\mathsf{ABKS.sk} \leftarrow \mathsf{ABKS.KeyGen}(\mathsf{ABKS.mk}, I_{\mathsf{KeyGen}})$, sets $\mathsf{sk} = (\mathsf{ABE.sk}, \mathsf{ABKS.sk})$, and sends sk to the data owner/user over an authenticated private channel.

$\underline{\mathsf{BuildIndex}}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$: The data owner runs $(\mathsf{Sig.sk}, \mathsf{Sig.pk}) \leftarrow \mathsf{Sig.KeyGen}(1^\ell)$, keeps Sig.sk private and makes 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 $\mathsf{SE.sk}_j \leftarrow \mathsf{SE.KeyGen}(1^\ell)$, $\mathsf{cph}_{\mathsf{SE}_j} \leftarrow \mathsf{SE.Enc}(\mathsf{SE.sk}_j, \mathsf{F}_j)$, 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 $\mathsf{KS}_i, 1 \le i \le l$, 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 $\sigma_w \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{cph}_w || \mathsf{string}(\{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\}))$, where $\mathsf{ID}_{\mathsf{F}_j}$ and $\mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}}$ are identities for identifying data file F_j and data ciphertext $\mathsf{cph}_{\mathsf{F}_j}$, respectively.
3) Generate a bloom filter, a bloom filter signature and a 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 $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{string}(\{\mathsf{cph}_w | w \in \mathsf{KS}_i\}))$.
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 $\mathsf{Au} = (\sigma, \sigma_1, \ldots, \sigma_l, \mathsf{cph}_{\mathsf{BF}_1}, \ldots, \mathsf{cph}_{\mathsf{BF}_l}, \sigma_{\mathsf{BF}_1}, \ldots, \sigma_{\mathsf{BF}_l}, \{\sigma_w | w \in \cup_{i=1}^{l} \mathsf{KS}_i\})$, $\mathsf{Index} = (\{\mathsf{cph}_w | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}, \{\mathsf{MP}(\mathsf{cph}_w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\})$ and $\mathsf{D}_{\mathsf{cph}} = (\{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{F}_j \in \mathsf{FS}\})$.

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

$\underline{\mathsf{SearchIndex}}(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}, \mathsf{tk})$: Let rslt be an empty set and $\mathsf{proof} = (\sigma)$ initially. The cloud enumerates $\prod_i = \{\mathsf{cph}_w | w \in \mathsf{KS}_i\}, 1 \le i \le l$, which are the keyword ciphertexts with respect to the same access control policy:

- For each $\mathsf{cph}_w \in \prod_i$, it runs $\gamma \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$. If $\gamma = 0$, then it continues to next keyword ciphertext in $\prod_i$; otherwise it adds the tuple $(\mathsf{cph}_w, \{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\})$ to rslt and $(\sigma_w, \mathsf{cph}_{\mathsf{BF}_i})$ to proof.
- If there exist no $\gamma = 1$ after enumerating all cph_w in $\prod_i$, then add $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i})$ to proof.

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

1) Verify 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 return 0; otherwise, continue to execute the following.
2) For $i = 1, \ldots, l$, it executes as follows to verify that the cloud indeed returned correct result for each group:
   $\underline{\text{Case 1:}}$ If $(\mathsf{cph}_w, \{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\}) \in \mathsf{rslt}$, where cph_w corresponds to the same access control policy as what is specified by $\mathsf{cph}_{\mathsf{BF}_i}$, then it runs $\gamma \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$ and $\gamma' \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_w, \mathsf{cph}_w || \mathsf{string}(\{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\}))$ to verify whether or not cph_w matches tk and all the associated data ciphertexts are returned by the cloud. If either $\gamma = 0$ or $\gamma' = 0$, then return 0, otherwise it continues to $i = i + 1$.
   $\underline{\text{Case 2:}}$ If $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i}) \in \mathsf{rslt}$, then it continues to verify integrity of the masked Bloom filter by running $\gamma' \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_{\mathsf{BF}_i}, \mathsf{BF}'_i || \mathsf{cph}_{\mathsf{BF}_i})$. If $\gamma' = 0$, return 0; otherwise it executes below:
   - If the data user is authorized, it computes $\mathsf{M} \leftarrow \mathsf{ABE.Dec}(\mathsf{ABE.sk}, \mathsf{cph}_{\mathsf{BF}_i})$, $\mathsf{BF}_i = H(\mathsf{M}) \bigotimes \mathsf{BF}'_i$. It executes $\delta \leftarrow \mathsf{BFVerify}(\{H'_1, \ldots, H'_k\}, \mathsf{BF}_i, w)$ to check whether w is present in the keyword group represented by BF_i.
     - If $\delta = 0$, meaning w is not present in the keyword group represented by BF_i, then it continues to $i = i + 1$.
     - If $\delta = 1$, it downloads $\prod_i = \{\mathsf{cph}_w | w \in \mathsf{KS}_i\}$ and σ_i from the cloud, and runs $\eta \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_i, \mathsf{string}(\{\mathsf{cph}_w | w \in \mathsf{KS}_i\}))$. If $\eta = 0$, it returns 0; otherwise it runs $\tau \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$ by enumerating cph_w in $\prod_i$. If there exists some $\tau = 1$ after enumerating all cph_w (meaning there exists some cph_w matches tk), it returns 0; otherwise it continues to $i = i + 1$.
   - If the data user is unauthorized, then it continues to $i = i + 1$ because $\mathsf{cph}_{\mathsf{BF}_i}$ cannot be decrypted.

   $\underline{\text{Case 3:}}$ Return 0 since the verification should be executed in either case above if the search result is correct.
3) Return 1 if all tuples in the search result have been verified, and 0 otherwise.

Fig. 3. VABKS construction

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 secure against chosen-plaintext attack, H is a secure pseudorandom generator and the ABKS is selectively secure against chosen keyword attack, the VABKS scheme is selectively secure 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 secure against chosen-plaintext attack, 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. We do not consider the asymptotic complexity of VABKS because it uses multiple building-blocks (e.g., the signature and ABE scheme) that can be instantiated with any secure ones. 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 [38].

We implemented ABKS and VABKS in JAVA based on the Java Pairing Based Cryptography library (jPBC) [38]. In our implementations, we instantiated the bilinear map with Type A pairing ($\ell = 512$), which offers a level of security that is equivalent to 1024-bit DLOG [38]. For both CP-VABKS and KP-VABKS, we instantiated the symmetric encryption scheme with AES-CBC, and the signature scheme with DSA signature scheme provided by JDK1.6. We instantiated ABKS, ABE with CP-ABKS, CP-ABE [25] for CP-ABKS, and KP-ABKS, KP-ABE [24] 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 complexities of the ABKS schemes. We observe that in the CP-ABKS scheme, the complexity of KeyGen is almost the same as that of Enc. In the KP-ABKS scheme, KeyGen is more expensive than Enc. In both schemes, the two Search algorithms incur almost the same cost.

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

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).

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 50 with step length 10. We ran each experiment for 10 times to obtain the average execution time. Table II shows the execution time of the two ABKS schemes.

KP-ABKS	KeyGen	0.088	0.786	1.539	2.316	3.081	3.863
	Enc	0.108	0.539	1.016	1.492	1.983	2.434
	TokenGen	0.073	0.331	0.627	0.917	1.211	1.504
	Search	0.049	0.275	0.480	0.711	0.947	1.182
	S(N)	1	10	20	30	40	50
CP-ABKS	KeyGen	0.107	0.686	1.275	1.901	2.525	3.151
	Enc	0.121	0.681	1.304	1.923	2.546	3.169
	TokenGen	0.088	0.349	0.673	0.932	1.228	1.513
	Search	0.061	0.329	0.493	0.728	0.97	1.202

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.

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, the keyword encryption algorithm is executed only once for each keyword, whereas the keyword search algorithm will be performed as many times as needed. Furthermore, we advocate the burden on the data users be further eased by outsourcing the keyword search operation 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 with real data, which 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 set $k = 28$ and $m = 10KB$ for Bloom filter so that $\frac{m}{n} = \frac{10*8*1024}{2019} \approx 40$ and the false-positive rate is around 4.5×10^{-9}. We vary the access control policy ranging from 1 to 50 attributes with step-length 10. In each experiment, we encrypted all keywords with the same access control policy. 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.

Figure 4(a) shows the execution time of BuildIndex that was run by the data owner. We observe that with the same attribute/policy complexity, CP-VABKS is more costly 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. We simulated that algorithm SearchIndex needs to conduct search operations over 1,010 keyword ciphertexts to find the matched keyword ciphertext. We observe that the execution time of TokenGen and Verify is really small, when 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 and auxiliary information, including 2,019 keyword ciphertexts, bloom filter and signatures. We also see that CP-VABKS consumes around two times more storage space than KP-VABKS with the same attribute/policy complexity. These discrepancies should serve as a factor when deciding whether to use CP-VABKS or KP-VABKS in practice.

VI. Conclusion

We have introduced a novel cryptographic solution called verifiable attribute-based keyword search, for secure cloud computing over outsourced encrypted data. The solution achieves the following: Data owners can control the search and use of their outsourced encrypted data according to their access control policies, while authorized data users can outsource the often costly search operations to the cloud and forces the cloud to faithfully execute the search operations. Performance evaluation shows that the new tool is practical. Our study focused on static data, and one interesting open problem for future research is to accommodate dynamic data.

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

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

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

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

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

[15] 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, 2009.

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

[17] 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.

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

[19] 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, 2011.

[20] 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, 2008.

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

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

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

[24] 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, 2006.

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

[26] 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.

[27] 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.

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

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

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

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

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

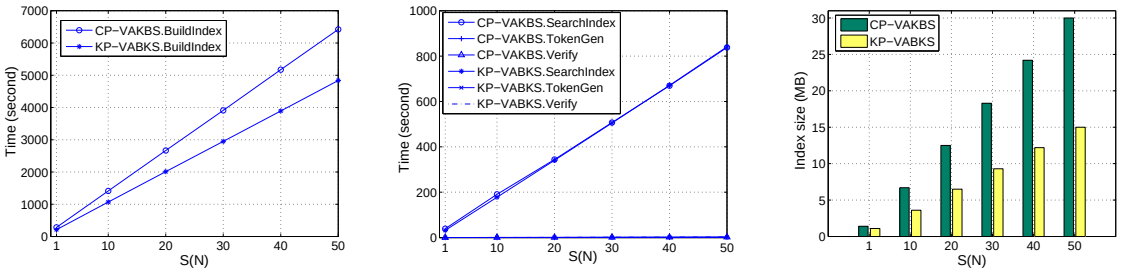
[36] R. Canetti, S. Halevi, and J. Katz, "Chosen-ciphertext security from identity-based encryption," in *EUROCRYPT*, pp. 207–222, 2004.

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

[38] "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



(a) BuildIndex (b) TokenGen, SearchIndex and Verify (c) Size of index and auxiliary information

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.

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 PolySat($\mathsf{T}_{v'}, \mathsf{Atts}^*, q_v(\mathsf{Index}(v'))$), where $q_v(\mathsf{Index}(v'))$ is known to the challenger;
  - otherwise, call 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 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 $W' = f^{r_1} = g^{cr_1}$, $W = Qf^{r_1 dH_2(w_\lambda)} = Qg^{r_1 cdH_2(w_\lambda)} = g^{a(r_1+r_2)}g^{br_1H_2(w_\lambda)}$, $W_0 = g^{r_2}$, 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 3. ∎

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

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

$g^{a(r_1+r_2)}$. 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. 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. ■