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 common nowadays for data owners to outsource their data to the cloud. Since the cloud cannot be fully trusted, the outsourced data should be encrypted. This however brings a range of problems, such as: How should a data owner grant search capabilities to the data users? How can the authorized data users search over a data owner’s outsourced encrypted data? How can the data users be assured that the cloud faithfully executed the search operations on their behalf? Motivated by these questions, we propose a novel cryptographic solution, called verifiable attribute-based keyword search (VABKS). The 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 search operations. We formally define the security requirements of VABKS and describe a construction that satisfies 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 low price. Despite the benefits, data outsourcing deprives data owners of direct control over their outsourced data. To alleviate concerns, data owners should encrypt their data before outsourcing to the cloud. However, encryption can hinder some useful functions such as searching over the outsourced encrypted data while enforcing an access control policy. Moreover, it is natural to outsource the search operations to the cloud, while keeping the outsourced data private. There is a need to allow the data users to verify whether the cloud faithfully executed the search operations or not. To the best of our knowledge, existing solutions cannot achieve these objectives simultaneously.

A. Our Contributions

We propose a novel cryptographic primitive, called verifiable attribute-based keyword search (VABKS). This primitive allows a data owner to control the search, and use of, its outsourced encrypted data according to an access control policy, while allowing the legitimate data users to outsource the (often costly) search operations to the cloud and verify whether or not the cloud has faithfully executed the search operations. In other words, a data user with proper credentials (corresponding to a data owner’s access control policy) can (i) search over the data owner’s outsourced encrypted data, (ii) outsource the search operations to the cloud, and (iii) verify whether or not the cloud has faithfully executed the search operations. We formally define the security properties of VABKS and present a scheme that provably satisfies them. The scheme is constructed in a modular fashion, by using attribute-based encryption, bloom filter, digital signature, and a new building-block we call attribute-based keyword search (ABKS) that may be of independent value. Experimental evaluation shows that the VABKS solutions are practical.

B. Related Work

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

Attribute-Based Encryption (ABE). ABE is a popular method for enforcing access control policies via cryptographic means. Basically, this technique allows entities with proper credentials to decrypt a ciphertext that was encrypted according to an access control policy [1]. Depending on how the access control policy is enforced, there are two variants: KP-ABE (key-policy ABE) where the decryption key is associated to the access control policy [2], and CP-ABE (ciphertext-policy ABE) where the ciphertext is associated to the access control policy [3]. ABE has been enriched with various features (e.g., [4]–[7]). In this paper, we use ABE to construct a new primitive called attribute-based keyword search (ABKS), by which keywords are encrypted according to an access control policy and data users with proper cryptographic credentials can generate tokens that can be used to search over the outsourced encrypted data. This effectively prevents a data owner from knowing the keywords a data user is searching for, while requiring no interactions between the data users and the data owners/trusted authorities. This is in contrast to [8], where the data users interact with the data owners/trusted authorities to obtain search tokens.

Keyword Search over Encrypted Data. This technique allows a data owner to generate some tokens that can be used by a data user to search over the data owner’s encrypted data. Existing solutions for keyword search over encrypted data can be classified into two categories: searchable encryption in the symmetric-key setting (e.g., [9]–[18]) and searchable encryption in the public-key setting (e.g., [8], [19]–[22]). Several variants (e.g., [23]–[26]) have been proposed to support complex search operations. Moreover, searchable encryption in the multi-users setting has been investigated as well [12], [27], where the data owner can enforce an access control policy by distributing some (stateful) secret keys to the authorized
users. However, all these solutions do not solve the problem we study, because (i) some of these solutions require interactions between the data users and the data owners (or a trusted proxy, such as a trapdoor generation entity [8]) to grant search capabilities, and (ii) all these solutions (except [18]) 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 operations 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 keyword search operations. This is true even for the powerful technique called predicate encryption [28], [29], which does not offer the desired verifiability.

**Verifiable Keyword Search.** Recently, verifiable keyword search solutions have been proposed in [30]–[32], where each keyword is represented as a root of some polynomial. It is possible to check whether a keyword is present by evaluating the polynomial on the keyword and verifying whether the output is zero or not. However, these approaches work only when keywords are sent in plaintext to the cloud, and are not suitable for our purpose because the cloud should not learn anything about the keywords. It is worth mentioning that the secure verifiable keyword search in the symmetric-key setting [18] 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 KP-ABKS and CP-ABKS schemes and analyzes their security properties. 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 \( \text{string}(S) \) denote the concatenation of elements of \( S \) ordered by their hash values. Let \( U = \{a_1, \ldots, a_n\} \) be a set of attributes that are used to specify access control policies.

**A. Cryptographic Assumption**

Let \( p \) be an \( \ell \)-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 \( Q = h^{r_1 + r_2} \) at most with a negligible advantage in security parameter \( \ell \), where “advantage” is defined as

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

**Generic Bilinear Group** [33]. Let \( \psi_0, \psi_1 \) be two random encodings of the additive group \( \mathbb{Z}_p^+ \) such that \( \psi_0, \psi_1 \) 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 \( \psi_0(1) \), \( g^a \) denote \( \psi_0(a) \), \( e(g, g) \) denote \( \psi_1(1) \), and \( e(g, g)^a \) denote \( \psi_1(a) \).

**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 \( \ell \)-bit seed and generates a \( m \)-bit string that cannot be distinguished from a \( m \)-bit random string by any polynomial-time algorithm (in \( \ell \)).

B. Bloom Filter for Membership Query

A Bloom filter [35] is a data structure for succinctly representing a static set, while allowing membership queries. A \( m \)-bit Bloom filter is an array of \( m \) bits, which are all initialized as 0. It uses \( k \) independent universal hash functions \( H'_1, \ldots, H'_k \) with the same 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, where \( 1 \leq j \leq k \). To determine whether \( w \) belongs to \( S \) or not, once can check whether all of the bits corresponding to \( H'_j(w) \) equal to 1, where \( 1 \leq j \leq k \). If not, it is certain that \( w \notin S \); otherwise, \( w \in S \) with a high probability (i.e., there is a non-zero false-positive rate). Suppose the hash functions are perfectly random and \( n \) elements are hashed into a \( m \)-bit Bloom filter, the false-positive rate is \( (1 - (1 - \frac{1}{m})^n)^k \approx (1 - e^{-kn/m})^k \). Note that \( k = (\ln 2)m/n \) hash functions lead to the minimal false-positive rate \((0.6185)^{m/n} \). A \( m \)-bit Bloom filter has two associated algorithms:

- \( BF \leftarrow BFGen(\{H'_1, \ldots, H'_k\}, \{w_1, \ldots, w_n\}) \): This algorithm generates a \( m \)-bit Bloom filter by hashing a data set \( S = \{w_1, \ldots, w_n\} \) with \( \{H'_1, \ldots, H'_k\} \).
- \( \{0,1\} \leftarrow BFVerify(\{H'_1, \ldots, H'_k\}, BF, w) \): This algorithm returns 1 if \( w \in S \), and 0 otherwise.

C. Access Trees for Representing Access Control Policies

Access trees can represent access control policies [2]. In an access tree, a leaf is associated with an attribute and an inner node represents a threshold gate. Let \( \text{num}_v \) be the number of children of node \( v \), and label the children from the left to the right as \( 1, \ldots, \text{num}_v \). Let \( k_v, 1 \leq k_v \leq \text{num}_v \), be the threshold value associated with node \( v \), where \( k_v = 1 \) represents the OR gate and \( k_v = \text{num}_v \) represents the AND gate. Let \( \text{parent}(v) \) denote the parent of node \( v \), \( \text{ind}(v) \) denote the label of node \( v \), \( \text{att}(v) \) denote the attribute associated to leaf node \( v \), \( \text{lv}(T_v) \) denote the set of leaves of access tree \( T_v \), and \( T_v \) denote the subtree of \( T \) rooted at node \( v \) (e.g., \( T_{root} = T \)).

Let \( F(\text{Atts}, T_v) = 1 \) indicate that an attribute set \( \text{Atts} \) satisfies the access control policy represented by subtree \( T_v \), where \( F(\text{Atts}, T_v) \) can be evaluated iteratively as follows:
• In the case $v$ is a leaf: If $\text{att}(v) \in \text{Atts}$, set $F(\text{Atts}, T_v) = 1$; otherwise, set $F(\text{Atts}, T_v) = 0$.
• In the case $v$ is an inner node with children $v_1, \ldots, v_{\text{num}_v}$:

  If there exists a subset $I \subseteq \{1, \ldots, \text{num}_v\}$ such that $|I| \geq k_v$ and $\forall j \in I, F(\text{Atts}, T_{v_j}) = 1$, set $F(\text{Atts}, T_v) = 1$; otherwise, set $F(\text{Atts}, T_v) = 0$.

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

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

This algorithm generates a polynomial $q_v$ of degree $k_v - 1$ for each node $v$ in a top-down fashion (for each leaf node $k_v = 1$):

- If $v$ is the root of $T$ (i.e., $v = \text{root}$), set $q_v(0) = s$ and randomly pick $k_v - 1$ coefficients for polynomial $q_v$.
- If $v$ is a leaf of $T$, set $q_v(0) = g_{\text{parent}(v)}(\text{ind}(v))$.
- If $v$ is an inner node (but not the root), set $q_v(0) = g_{\text{parent}(v)}(\text{ind}(v))$ and randomly select $k_v - 1$ coefficients for polynomial $q_v$.

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

Given an access tree $T$ and a set of values $\{E_{u_1}, \ldots, E_{u_m}\}$, where $u_1, \ldots, u_m$ are the leaves of $T$, $F(\{\text{att}(u_1), \ldots, \text{att}(u_m)\}, T) = 1$, $E_{u_j} = e(g, h)^{q_{u_j}(0)}$ 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 $T$, the algorithm for reconstructing $e(g, h)^s$ is denoted by

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

This algorithm executes the following steps with respect to node $v$ in a bottom-top fashion according to $T$:

• If $F(\{\text{att}(u_1), \ldots, \text{att}(u_m)\}, T_v) = 0$, then set $E_v = 1$.
• If $F(\{\text{att}(u_1), \ldots, \text{att}(u_m)\}, T_v) = 1$, then execute the following:
  - 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 $j$.
  - If $v$ is an inner node (including the root), for $v$’s children nodes $\{v_1, \ldots, v_{\text{num}_v}\}$, there exists a set of indices $S$ such that $|S| = k_v$, $j \in S$, and $F(\{\text{att}(u_1), \ldots, \text{att}(u_m)\}, T_{v_j}) = 1$. Set

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

    where $\Delta_v = \prod_{j \in S, i \neq j} \Delta_{v_j}$.

When the algorithm halts, the root of $T$ is associated with

$$E_{\text{root}} = e(g, h)^{\text{secret}(0)} = e(g, h)^s.$$  

### III. Attribute-Based Keyword Search (ABKS)

This new primitive allows a data owner to specify a policy for controlling the keyword search operations over its outsourced encrypted data. That is, a data user who possesses attributes that satisfy the data owner’s policy can conduct keyword search over the outsourced encrypted data. This primitive naturally has two variants: KP-ABKS (key-policy ABKS) where the cryptographic credentials are associated to the access control policy, and CP-ABKS (ciphertext-policy ABKS) where the ciphertext is associated to the access control policy. To unify the presentation, let $I_{\text{Enc}}$ denote the input to encryption function $\text{Enc}$ and $I_{\text{KeyGen}}$ denote the input to key generation function $\text{KeyGen}$. For CP-ABKS, $I_{\text{Enc}}$ and $I_{\text{KeyGen}}$ are respectively the access tree and the attribute set; for KP-ABKS, $I_{\text{Enc}}$ and $I_{\text{KeyGen}}$ are respectively the access set and the access tree. Let $F(I_{\text{KeyGen}}, I_{\text{Enc}}) = 1$ denote $I_{\text{KeyGen}}$ satisfies $I_{\text{Enc}}$ in CP-ABKS and $I_{\text{Enc}}$ satisfies $I_{\text{KeyGen}}$ in KP-ABKS.

### A. Definition and Security

The model of ABKS is: A data owner outsources its encrypted keywords to the cloud, a data user generates search tokens according to some keywords, and the cloud, who receives search tokens from the user, conducts the search operations over outsourced encrypted keywords.

**Definition 1:** ABKS consists of the following algorithms:

- $(mk, pm) \leftarrow \text{Setup}(1^n)$: This algorithm initializes the public parameter $pm$ and generates a master key $mk$.
- $sk \leftarrow \text{KeyGen}(mk, I_{\text{KeyGen}})$: This algorithm outputs credential $sk$ for a user according to $I_{\text{KeyGen}}$.
- $\text{cph} \leftarrow \text{Enc}(w, I_{\text{Enc}})$: This algorithm encrypts keyword $w$ to obtain ciphertext $cph$.
- $tk \leftarrow \text{TokenGen}(sk, w)$: This algorithm allows a data user to generate a search token $tk$ according to its credential $sk$ and keyword $w$.
- $\{0, 1\} \leftarrow \text{Search}(\text{cph}, tk)$: This algorithm returns 1 if (i) $F(I_{\text{KeyGen}}, I_{\text{Enc}}) = 1$ and (ii) ciphertext $cph$ and token $tk$ correspond to the same keyword, and return 0 otherwise.

An ABKS scheme is correct if the following holds: Given $(mk, pm) \leftarrow \text{Setup}(1^n)$, $sk \leftarrow \text{KeyGen}(mk, I_{\text{KeyGen}})$ and $F(I_{\text{KeyGen}}, I_{\text{Enc}}) = 1$, $\text{cph} \leftarrow \text{Enc}(w, I_{\text{Enc}})$ and $tk \leftarrow \text{TokenGen}(sk, w)$, $\text{Search}(\text{cph}, tk)$ always returns 1.

The adversary model against ABKS is the following: data owners and authorized data users are trusted, but the cloud is trusted but curious (i.e., executing the protocol honestly but attempting to infer private information as well). Intuitively, security means that the cloud learn nothing beyond the search results. Specifically, given a probabilistic polynomial-time adversary $\mathcal{A}$ (modeling the cloud), an ABKS scheme is secure if the following holds:

- **Selective security against chosen-keyword attack:** Without being given any matching search token, $\mathcal{A}$ cannot infer any information about the plaintext keyword of a keyword ciphertext in the selective security model, where $\mathcal{A}$ must determine $I_{\text{Enc}}$ it intends to attack before the system is bootstrapped [36]. We formalize this security property via the selective chosen-keyword attack game.
- **Keyword secrecy:** In the public-key setting, it is impossible to protect the search tokens (aka. predicate privacy [37]) against the keyword guessing attack. This is because $\mathcal{A}$ can encrypt a keyword of its choice and check whether the resulting keyword ciphertext and the target token
correspond to the same keyword, which is caused by the use of “deterministic encryption.” Therefore, we use a weaker security notion called keyword secrecy, assuring that the probability $A$ learning the keyword from the keyword ciphertext and search tokens is negligibly more than the probability of correct random keyword guess. We formalize this security property via the keyword secrecy game.

**Selectively Chosen-Keyword Attack (SCKA) Game:**

**Setup:** $A$ selects a non-trivial challenge $\iota_{\text{Enc}}$ (a trivial challenge $\iota_{\text{Enc}}$ 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 Setup($1^\ell$) to generate the public parameter $\text{pm}$ and the master key $\text{mk}$.

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

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

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

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

**Guess:** $A$ outputs a bit $\lambda'$, and wins the game if $\lambda' = \lambda$.

Let $\Pr[\lambda' = \lambda'] - \frac{1}{2}$ be the advantage of $A$ 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 $A$ winning the SCKA game is negligible in security parameter $\ell$.

**Keyword Secrecy Game:**

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

**Phase 1:** $A$ can query the following oracles for polynomially many times:

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

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

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

**Definition 3:** An ABKS scheme achieves keyword secrecy if the probability that $A$ wins the keyword secrecy game is at most $\frac{1}{2} + \epsilon$, where $\mathcal{M}$ is the keyword space, $q$ is the number of distinct keywords that the adversary has attempted, and $\epsilon$ is a negligible in security parameter $\ell$.

**B. Construction**

The basic idea underlying the construction is the following: each keyword ciphertext and each search token has two parts, one is associated to the keyword and the other is associated to the attributes (or access control policy). If the attributes satisfy the access control policy, one can determine whether the search token and keyword ciphertext correspond to the same keyword or not. Consider KP-ABKS as an example. Let $H_1 : \{0, 1\}^* \rightarrow G$ be a hash function modeled as random oracle and $H_2 : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ be an one-way hash function. A data user’s credentials are generated by letting $t \leftarrow \mathbb{Z}_p^*$, $A_v = g^{ac}(0)H_1(\text{att}(v))^t, B_v = g^t$ for each leaf $v$, where $g$ is a 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” $w$ with randomness $r_1, r_2 \leftarrow \mathbb{Z}_p$, by letting $W' = g^{r_1}, W = g^{a(r_1+r_2)}g^{bH_2(w)r_1}$ and $W_0 = g^{r_2}$ where $g^a, g^b, g^c \in G$ are public keys, and the other is associated to attribute set $\text{Atts}$ by letting $W_j = H_1(\text{att}(v))^{t_j}$ for each at $j \in \text{Atts}$. The two parts are tied together via $t_2$.

- Given a set of credentials, a search token for keyword $w$ is generated with two parts: one is associated to $w$ as $\text{tok}_1 = (g^ag^{bH_2(w)})^s$ and $\text{tok}_2 = g^s$ for some $s \leftarrow \mathbb{Z}_p^*$, and the other is associated to the credentials by letting $A_v' = A_v, B_v' = B_v$ for each $v \in \text{lv}(T)$. The two parts are tied together via randomness $s$.

If the attribute set $\text{Atts}$ satisfies the access tree $T$, the cloud can use $A'_v, B'_v$ and $W_0, W_j$ to recover $e(g, g^ac)^{r_2}$, which can be used to test the keyword equality as elaborated below.

1) KP-ABKS Construction and Security Analysis: Let $\ell$ be the primary security parameter. It consists of the following algorithms.

**Setup($1^\ell$):** Select a bilinear map $e : G \times G \rightarrow G_T$, where $G$ and $G_T$ are cyclic groups of order $p$, which is an $\ell$-bit prime. Let $H_1 : \{0, 1\}^* \rightarrow G$ be a hash function modeled as random oracle and $H_2 : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ be an one-way hash function, select $a, b, c \leftarrow \mathbb{Z}_p^*$ and $g \leftarrow G$, and set $\text{pm} = (H_1, H_2, e, g, p, g^a, g^b, g^c, G, G_T), \text{mk} = (a, b, c)$.

$\text{KeyGen}(\text{mk}, T)$: Execute Share($T$, $ac$) to obtain secret share $q_v(0)$ of $ac$ for each leave $v \in \text{lv}(T)$ on access tree $T$. For each leaf $v \in \text{lv}(T)$, pick $t \leftarrow \mathbb{Z}_p^*$, and compute $A_v = g^{q_v(0)H_1(\text{att}(v))^t}$ and $B_v = g^t$. Set $\text{sk} = (T, \{(A_v, B_v) | v \in \text{lv}(T)\})$. 

Enc(w, Atts): Select $r_1, r_2 \leftarrow \mathbb{Z}_p$, and compute $W' = g^{r_1}$, $W = g^{a(r_1 + r_2)}h^{H_2(w)r_1}$ and $W_0 = g^{r_2}$. For each $a_j \in$ Atts, compute $W_j = H_1(a_j)^{r_2}$. Set

$$\text{cph} = (\text{Atts}, W, W', W_0, \{W_j|a_j \in \text{Atts}\}).$$

TokenGen($sk, w$): Select $s \leftarrow \mathbb{Z}_p$, and compute $A'_s = A_s \ast B'_s = B_s \ast$ for each $v \in \ell v s(T)$. Compute $\text{tok}_1 = (g^a g^{bH_2(w)})^s$, and $\text{tok}_2 = g^{cs}$. Set

$$\text{tk} = (\text{tok}_1, \text{tok}_2, T, \{(A'_s, B'_s)|v \in \ell v s(T)\})$$

Search($tk, \text{cph}$): Given an attribute set Atts specified in $\text{cph}$, select an attribute set $S$ satisfying the access tree $T$ specified in $tk$. If $S$ does not exist, return 0; otherwise, for each $a_j \in$ Atts, compute $E_v = e(A'_s, W_0)/e(B'_s, W_j) = e(g, g)^{s2q_2}$, where $\text{att}(v) = a_j$ for $v \in \ell v s(T)$. Compute $e(g, g)^{s2q_2}$ $\leftarrow$ $\text{Combine}(T, \{E_v|\text{att}(v) \in S\})$ so that

$$\text{E}_{\text{root}} = e(g, g)^{ac}.$$ Return 1 if $e(W', \text{tok}_1)\text{E}_{\text{root}} = e(W, \text{tok}_2)$, and 0 otherwise.

The scheme is correct because

\[
e(W', \text{tok}_1)\text{E}_{\text{root}} = e(g^{cr_1}, g^{cr_2}h^{H_2(w)})^s\text{E}_{\text{root}} = e(g, g)^{ac}r_2h^{H_2(w)}r_1
\]

\[
e(W, \text{tok}_2) = e(g^{a(r_1 + r_2)}h^{H_2(w)r_1}, g^{cs}) = e(g, g)^{ac}h^{H_2(w)r_1}
\]

The scheme is secure because of the following theorems, whose proofs are given in Appendix A and Appendix B, respectively.

**Theorem 1:** Given the DL assumption and one-way hash function $H_2$, 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 in the random oracle model.

2) CP-ABKS Construction and Security Analysis: Let $\ell$ be the primary security parameter. It consists of the following algorithms.

Setup($\ell$): Select a bilinear group $e : G \times G \rightarrow G_T$, where $G$ and $G_T$ are cyclic groups of order $p$, which is an $\ell$-bit prime. Let $H_1 : \{0, 1\}^* \rightarrow G$ be a hash function modeled as random oracle and $H_2 : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ be an one-way hash function, select $a, b, c \leftarrow \mathbb{Z}_p$ and $g \leftarrow G$, and set

$$\text{pm} = (H_1, H_2, g, p, a, b, g^c, G, G_T), \text{mk} = (a, b, c).$$

KeyGen($\text{mk, Atts})$: Select $r \leftarrow \mathbb{Z}_p$, compute $A = g^{(ac - r)/b}$. For each $a_j \in$ Atts, select $r_j \leftarrow \mathbb{Z}_p$, and computes $A_j = g^r H_1(a_j)^{r_j}$ and $B_j = g^{r_j}$. Set

$$\text{sk} = (\text{Atts}, A, \{(A_j, B_j)|a_j \in \text{Atts}\}).$$

Enc($w, T$): Select $r_1, r_2 \leftarrow \mathbb{Z}_p$, and compute $W = g^{r_1}$, $W_0 = g^{a(r_1 + r_2)}h^{H_2(w)r_1}$ and $W' = g^{r_2}$. Compute secret shares of $r_2$ for each leave of access tree $T$ as $\{q_v(0)|v \in \ell v s(T)\} \leftarrow \text{Share}(T, r_2)$. For each $v \in \ell v s(T)$, compute $W_v = g^{r_2(0)}$ and $D_v = H_1(\text{att}(v))q_v(0)$. Set

$$\text{cph} = (T, W, W_0, W', \{W_v, D_v|v \in \ell v s(T)\}).$$

TokenGen($sk, w$): Select $s \leftarrow \mathbb{Z}_p$, and compute $\text{tok}_1 = (g^a g^{bH_2(w)})^s$, $\text{tok}_2 = g^{cs}$ and $\text{tok}_3 = A^2 = g^{(ac - r)/b}$. For each $a_j \in$ Atts, compute $A_j = A_j^2$ and $B_j = B_j^2$. Set

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

Search($tk, \text{cph})$: Given an attribute set Atts as specified in $tk$, select an attribute set $S$ that satisfies the access tree $T$ specified in $\text{cph}$. If $S$ does not exist, return 0; otherwise, for each $a_j \in$ Atts, compute $E_v = e(A'_j, W_0)/e(B'_j, W_j) = e(g, g)^{s2q_2}$, where $\text{att}(v) = a_j$ for $v \in \ell v s(T)$. Compute $e(g, g)^{s2q_2}$ $\leftarrow$ $\text{Combine}(T, \{E_v|\text{att}(v) \in S\})$ and $E_{\text{root}} = e(g, g)^{s2r_2}$. Return 1 if $e(W_0, \text{tok}_2) = e(W, \text{tok}_1)E_{\text{root}}(\text{tok}_3, W')$, and 0 otherwise.

Correctness of the scheme can be verified similarly to that of KP-ABKS. Security of the scheme is assured by the following theorems. The proof of the former one is deferred to Appendix C, and the proof of the latter one is omitted because it is similar to that of Theorem 2.

**Theorem 3:** Given the one-way hash function $H_2$, the 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 in the random oracle model.

IV. VERIFIABLE ATTRIBUTE-BASED KEYWORD SEARCH

In the model of ABKS, the party (e.g., cloud) is assumed to execute the search operation faithfully (despite that the party may attempt to infer useful information about the keywords). VABKS achieves the goal of ABKS despite that the party executing the search operation may be malicious.

A. Model

We consider the system model illustrated in Figure 1, which involves four parties: a data owner, who outsources its encrypted data as well as encrypted keyword-index to the cloud; a cloud, which provides storage services and can conduct keyword search operations on behalf of the data users; a data user, who is to retrieve the data owner’s encrypted data according to some keyword (i.e., keyword search); a trusted authority, which issues credentials to the data owners/users. The credentials are sent over authenticated private channels (which can be achieved through another layer of mechanisms).

![Fig. 1. VABKS system model](image-url)
The data owners are naturally trusted. Both authorized and unauthorized data users are semi-trusted, meaning that they may try to infer some sensitive information of interest. The cloud is not trusted as it may manipulate the search operations, which already implies that the cloud may manipulate the outsourced encrypted data.

B. Definition

Let $FS = \{F_1, \ldots, F_n\}$ be a set of data files. Let $KG_j$, $1 \leq j \leq l$, be a set of keywords (also called “keyword group”) that are encrypted with the same access control policy (i.e., access tree). Let $KG = \{KG_1, \ldots, KG_l\}$. For each keyword $w$, let $MP(w)$ be the set of identifiers identifying data files that contain keyword $w$. Let $MP = \{MP(w) | w \in \cup_{j=1}^{l} KG_j\}$. Let $D = (KG, MP, FS)$ denote keyword-index and the data files.

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

- $(mk, pm) \leftarrow Init(1^\ell)$: This algorithm is run by the trusted authority to initialize the system.
- $sk \leftarrow KeyGen(mk, I_{KeyGen})$: This algorithm is run by the trusted authority to issue credentials sk for data users/owners.
- $(Au, Index, D_{cph}) \leftarrow BuildIndex(\{I_{Enc}\}_1, \{I_{Enc}\}_n, D)$: This algorithm is run by a data owner to encrypt $D = (KG, MP, FS)$ to data ciphertext $D_{cph}$, index ciphertext $Index$ and auxiliary information $Au$, where $\{I_{Enc}\}_i$ is the set of access control policies respectively for encrypting the $l$ keyword groups $KG_1, \ldots, KG_l$, and $\{I_{Enc}'\}_n$ is the set of access control policies respectively for encrypting the $n$ data files $F_1, \ldots, F_n$ (it may happen that the access control policies for keywords and their respective data files are different).
- $tk \leftarrow TokenGen(sk, w)$: This algorithm is run by an authorized data user to generate a search token tk for keyword $w$.
- $(proof, rslt) \leftarrow SearchIndex(Au, Index, D_{cph}, tk)$: This algorithm is run by the cloud to conduct the search operations over encrypted index $Index$ on behalf of a data user. It outputs the search result rslt and a proof proof.
- $\{0, 1\} \leftarrow Verify(sk, w, tk, rslt, proof)$: This algorithm is run by the data user to verify that $(rslt, proof)$ is valid with respect to search token tk.

A VABKS scheme is correct if the following holds: given $(mk, pm) \leftarrow Init(1^\ell)$, sk $\leftarrow KeyGen(mk, I_{KeyGen})$, $(Au, Index, D_{cph}) \leftarrow BuildIndex(\{I_{Enc}\}_1, \{I_{Enc}'\}_n, D)$, $tk \leftarrow TokenGen(sk, w)$ and $(proof, rslt) \leftarrow SearchIndex(Au, Index, D_{cph}, tk)$, Verify$(sk, w, tk, rslt, proof)$ always returns $1$.

Informally, security of VABKS is defined as follows.

- **Data secrecy**: Given encrypted keywords and search tokens, $A$ still cannot learn any information (in a computational sense) about the encrypted data files. This definition can be formalized by the chosen-plaintext security game, where two challenges $D_0 = (KG, MP, FS_0)$, $D_1 = (KG, MP, FS_1)$ correspond to the same $KG$ and $MP$, and $|FS_0| = |FS_1|$.

- **Selective security against chosen-keyword attack**: Without seeing corresponding search tokens, $A$ cannot infer any information about the keyword from the keyword ciphertext. This property is extended from the selective security against chosen-keyword attack of ABKS.

- **Keyword secrecy**: Given encrypted data files, the probability that $A$ learns the plaintext keyword from the keyword ciphertext as well as the search tokens is no more than that of a random guess. This property is extended from the keyword secrecy of ABKS.

- **Verifiability**: If $A$ returns an incorrect search result, it can be detected by the user with an overwhelming probability. We formalize this security property via the following verifiability game.

Verifiability Game:

**Setup**: The challenger runs $(pm, mk) \leftarrow Init(1^\ell)$. $A$ selects $D = (KG, MP, FS), \{I_{Enc}\}_1$ and $\{I_{Enc}\}_n$ and sends them to the challenger. The challenger runs $(Au, Index, D_{cph}) \leftarrow BuildIndex(\{I_{Enc}\}_1, \{I_{Enc}\}_n, D)$, and gives $(Au, Index, D_{cph})$ to $A$.

**Phase 1**: $A$ can query the following oracles for polynomially many times.

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

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

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

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

C. Construction

A trivial solution for achieving verifiability is that a data user downloads the keyword ciphertexts and conduct the search operations locally. This solution incurs prohibitive communication and computational overhead. As highlighted in Figure 2, we instead let a data user outsource the keyword search operation to the cloud, and then verify that the cloud faithfully performed the keyword search operation. More specifically, the data owner uses the signatures and bloom filters as follows:

- **A keyword signature** is generated for each keyword ciphertext and its associated data ciphertexts. It is used
for preventing the cloud from returning incorrect data ciphertexts as the search result.

- For each keyword group, one bloom filter is built from its keywords. This allows a data user to check that the searched 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. A bloom filter signature is generated for the masked bloom filter and the random number ciphertext for assuring their integrity.

- A global signature is obtained by signing random number ciphertexts of all groups. It allows a data user to verify the integrity of the random number ciphertexts.

- A local signature is generated for all keyword ciphertexts within the same keyword group KGi. This signature allows the user to validate the integrity of keyword ciphertexts within the keyword group.

Figure 3 describes the VABKS scheme, which uses a signature scheme Sig = (KeyGen, Sign, Verify), a symmetric encryption scheme SE = (KeyGen, Enc, Dec), an ABE scheme ABE = (Setup, KeyGen, Enc, Dec), where the latter two encryption schemes are used to encrypt data files. The VABKS scheme is built on top of an ABKS scheme ABKS = (Setup, KeyGen, Enc, TokenGen, Search), which encrypts the keywords. Note that ABE and ABKS can be their ciphertext-policy variant or their key-policy variant, but for the same type. This leads to two variants of VABKS.

Note that in the Verify algorithm of Figure 3, when an authorized data user verifies a null search result for keyword group \{cphw|w ∈ KGi\}, where the user searches keyword w', it can happen that 1 ← BFVerify(H'1, ..., H'k, BF1, w') due to the false-positive of the Bloom filter. To validate the search result in this case, the Verify algorithm has to download \{cphw|w ∈ KGi\}, and checks the keyword ciphertexts one by one. We stress that this does not incur significant communication cost on average because we can set the false-positive rate to be as low as possible by choosing appropriate m and k (i.e., upon one search request, the “wasted” bandwidth communication and computational 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

Security of the VABKS scheme can be proven as the following theorems, whose proofs are deferred to Appendix D.

Theorem 5: If ABE and SE are secure against the chosen-plaintext attack, the VABKS scheme achieves the data secrecy.

Theorem 6: If ABE is secure against chosen-plaintext attack, H is a secure pseudorandom generator and ABKS is selectively secure against chosen keyword attack, the VABKS scheme is selectively secure against chosen-keyword attack.

Theorem 7: If ABE is secure against chosen-plaintext attack, H is a secure pseudorandom generator and ABKS achieves keyword secrecy, the VABKS scheme achieves keyword secrecy.

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

V. PERFORMANCE EVALUATION

We evaluate the efficiency of the ABKS schemes in terms of both asymptotic complexity and actual execution time, and the 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., signing and ABE schemes) that can be instantiate with any secure solutions. Asymptotic complexity is measured in terms of four kinds of operations: H1 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 E2 denotes the exponentiation operation in G2. We ignore multiplication and hash operations (other than H1) because they are much more efficient than the above operations [38].

We implemented ABKS and VABKS in JAVA, while using the Java Pairing Based Cryptography library (jPBC) [38].
Init(ℓ): Given security parameter ℓ, the attribute authority chooses k universal hash functions $H_1, \ldots, H_k$, which are used to construct a m-bit Bloom filter. Let $H : \{0, 1\}^\ell \rightarrow \{0, 1\}^m$ 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. This algorithm executes (ABE.pm, ABE.mk) ← ABE.Setup(1^\ell) and (ABKS.pm, ABKS.mk) ← ABKS.Setup(1^\ell). It sets the public parameters as pm = (ABE.pm, ABKS.pm, $H_1, \ldots, H_k$) and mk = (ABE.mk, ABKS.mk).

KeyGen(mk, I_{KeyGen}): The attribute authority runs ABE.sk ← ABE.KeyGen(ABE.mk, I_{KeyGen}) and ABKS.sk ← ABKS.KeyGen(ABKS.mk, I_{KeyGen}), sets sk = (ABE.sk, ABKS.sk), and sends sk to a data owner/user over an authenticated private channel.

BuildIndex(I_{Enc}, I_{Enc} n, D): The data owner runs $\text{Sig} \leftarrow \text{Sig.KeyGen}(1^\ell)$, keeps Sig.sk private and makes Sig.pk public. Given D = (KG = {KG_1, \ldots, KG_l}, MP = {MP(w) | w \in \cup_{i=1}^l KG_i}, FS = {F_1, \ldots, F_n}), the data owner executes as follows:

1) Encrypt each data file with hybrid encryption: $\forall F_j \in FS$, generate ciphertext $cph_{F_j} = (cph_{sk_j}, cph_{SE})$ by running $SE.sk_j \leftarrow SE.KeyGen(1^\ell)$, $cph_{SE} \leftarrow SE.Enc(SE.sk_j, F_j)$, and $cph_{sk_j} \leftarrow ABE.Enc(I_{Enc_j}, F_j)$.

2) Encrypt each keyword and generate a public signature: Given $KG_i, 1 \leq i \leq l$, for each $w \in KG_i$, run $cph_w \leftarrow ABE.Enc(I_{Enc_i}, w)$, set $MP(cph_w) = \{ID_{cph_F} | ID_{F_j} \in MP(w)\}$, and generate $\sigma_w \leftarrow \text{Sig.Sign}(\text{Sig.sk}, cph_w || \text{string}(\{cph_{F_j} | ID_{cph_{F_j}} \in MP(cph_w)\}))$, where $ID_{F_j}$ and $ID_{cph_{F_j}}$ are identifiers for identifying data file $F_j$ and data ciphertext $cph_{F_j}$, respectively.

3) Generate a bloom filter, a bloom filter signature and a local signature for each group $KG_i$: Let $BF_i \leftarrow BFGen(\{H_1, \ldots, H_k\}, KG_i)$, $cph_{BF_i} \leftarrow ABE.Enc(I_{Enc_i}, M)$ for some randomly chosen $M$ from the message space of ABE, compute $BF'_i = H(M) \otimes BF_i$ and generate $\sigma_{BF_i} \leftarrow \text{Sig.Sign}(\text{Sig.sk}, BF'_i || cph_{BF_i})$. Let $\sigma_i \leftarrow \text{Sig.Sign}(\text{Sig.sk}, \text{string}(\{cph_{w} | w \in KG_i\}))$.

4) Generate the global signature: Set $\sigma = \text{Sig.Sign}(\text{Sig.sk}, cph_{BF_1} \ldots || cph_{BF_l})$.

5) Let $Au = (\{\sigma_1, \ldots, \sigma_l, cph_{BF_1}, \ldots, cph_{BF_l}, \sigma_{BF_1}, \ldots, \sigma_{BF_l} \mid \{cph_{w} | w \in \cup_{i=1}^l KG_i\}$, Index = $\{\{cph_{w} | w \in \cup_{i=1}^l KG_i\}, \{MP(cph_{w}) | w \in \cup_{i=1}^l KG_i\}$ and $D_{cph} = \{\{cph_{F_j} | F_j \in FS\}$.

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

SearchIndex(Au, Index, D_{cph}, tk): Let rslt be an empty set and proof = $\sigma$ initially. The cloud enumerates $\prod_i = \{cph_{w} | w \in KG_i\}, 1 \leq i \leq l$, which are the keywords ciphertexts with respect to the same access control policy.

- For each $cph_{w} \in \prod_i$, it runs $\gamma \leftarrow ABKS.Search(cph_{w}, tk)$. If $\gamma = 0$, it continues to process the next keyword ciphertext in $\prod_i$; otherwise, it adds the tuple $(cph_{w} || ID_{cph_{BF_i}} \in MP(cph_{w}))$ to rslt and $(\sigma_w, cph_{BF_i})$ to proof.
- If there exist no $\gamma = 1$ after processing all $cph_{w} \in \prod_i$, then its adds $(BF'_i, cph_{BF_i}, \sigma_{BF_i})$ to proof.

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

1) Verify the integrity of the random number ciphertexts: Let $\gamma = \text{Sig.Verify}(\text{Sig.pk}, \sigma_1 || \ldots || cph_{BF_l})$. If $\gamma = 0$, 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 the correct result for each keyword group $i$:

   Case 1: If $(cph_{w} || ID_{cph_{BF_i}} \in MP(cph_{w})) \in rslt$, meaning there exists the keyword ciphertext $cph_{w}$, which corresponds to the same access control policy as what is specified by $cph_{BF_i}$, having the same key specified by $tk$, then it runs $\gamma \leftarrow ABKS.Search(cph_{w}, tk)$ and $\gamma' \leftarrow \text{Sig.Verify}(\text{Sig.pk}, \sigma_{BF_i}, cph_{w} || \text{string}(\{cph_{F_j} | ID_{cph_{F_j}} \in MP(cph_{w})\}))$ 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, continue to $i = i + 1$.

   Case 2: If $(BF'_i, cph_{BF_i}, \sigma_{BF_i}) \in proof$ meaning that there is no matching keyword ciphertext, then it continues to verify the integrity of the masked Bloom filter by running $\gamma' \leftarrow \text{Sig.Verify}(\text{Sig.pk}, \sigma_{BF_i}, BF'_i || cph_{BF_i})$. If $\gamma' = 0$, return 0; otherwise, execute the following:

   - If the data user is authorized, compute $M \leftarrow ABE.Dec(ABE.sk, cph_{BF_i})$, $BF_i = H(M) \otimes BF'_i$. Execute $\delta \leftarrow BFVerify(\{H_1, \ldots, H_k\}, BF_i, w)$ to check whether $w$ or not is present in the keyword group as represented by $BF_i$.
     - If $\delta = 0$, meaning that $w$ is not present in the keyword group as represented by $BF_i$, then continue to $i = i + 1$.
     - If $\delta = 1$, download $\prod_i = \{cph_{w} | w \in KG_i\}$ and $\sigma_i$ from the cloud, and run $\eta \leftarrow \text{Sig.Verify}(\text{Sig.pk}, \sigma_i$, string($\{cph_{w} | w \in KG_i\}$)). If $\eta = 0$, return 0; otherwise, run $\tau \leftarrow ABKS.Search(cph_{w}, tk)$ by enumerating $cph_{w}$ in $cph_{w} \in KG_i$. If there exists some $\tau = 1$ after processing all $cph_{w}$ (meaning that there exists some $cph_{w}$ that matches $tk$), return 0; otherwise, continue to $i = i + 1$.

   - If the data user is unauthorized, then it continues to $i = i + 1$ because $cph_{BF_i}$ cannot be decrypted.

   Case 3: If none of the above two cases happens, return 0.

3) Return 1 if all tuples in the search result have been verified, and 0 otherwise.
In our implementation, the bilinear map is instantiated as Type A pairing (\( \ell = 512 \)), which offers a level of security equivalent to 1024-bit DLOG [38]. For both CP-ABKS and KP-ABKS, we instantiated the symmetric encryption scheme as AES-CBC, and the signature scheme with DSA provided by JDK1.6. We instantiated ABKS, ABE as CP-ABKS, CP-ABE [3] for CP-ABKS, and KP-ABKS, KP-ABE [2] for KP-ABKS, respectively. Finally, we set the example access control policy as “\( \text{at}_1 \) AND \ldots AND \text{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.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Complexity</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>KP-ABKS</td>
<td>( 3N + N + H_1 )</td>
<td>( 2N + (2) )</td>
</tr>
<tr>
<td>ABKS</td>
<td>( (S + 4)E + S + H_1 )</td>
<td>( (S + 3)(2) )</td>
</tr>
<tr>
<td>TokenGen</td>
<td>( (2N + 2)E )</td>
<td>( (2N + 2)(2) )</td>
</tr>
<tr>
<td>Search</td>
<td>( (2N + 2)\text{Pair} + N + H_1 )</td>
<td>( (2N + 3)(2) )</td>
</tr>
<tr>
<td>CP-ABKS</td>
<td>( (2S + 2)E + S + H_1 )</td>
<td>( (2S + 1)(2) )</td>
</tr>
<tr>
<td>ABKS</td>
<td>( (2S + 4)E + N + H_1 )</td>
<td>( (2S + 3)(2) )</td>
</tr>
<tr>
<td>TokenGen</td>
<td>( (2S + 4)E )</td>
<td>( (2S + 3)(2) )</td>
</tr>
<tr>
<td>Search</td>
<td>( (2S + 4)\text{Pair} + N + L )</td>
<td>( (2S + 3)(2) )</td>
</tr>
</tbody>
</table>

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).

Actual Performance of the ABKS Schemes. 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>
<thead>
<tr>
<th>Scheme</th>
<th>Attribute Complexity</th>
<th>Execution Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KP-ABKS</td>
<td>( \text{KeyGen} ) 0.088</td>
<td>1.939</td>
</tr>
<tr>
<td>ABKS</td>
<td>( \text{Enc} ) 0.108</td>
<td>1.016</td>
</tr>
<tr>
<td>TokenGen</td>
<td>0.093</td>
<td>0.480</td>
</tr>
<tr>
<td>Search</td>
<td>0.939</td>
<td>0.581</td>
</tr>
<tr>
<td>CP-ABKS</td>
<td>( \text{KeyGen} ) 0.107</td>
<td>1.295</td>
</tr>
<tr>
<td>ABKS</td>
<td>( \text{Enc} ) 0.121</td>
<td>1.304</td>
</tr>
<tr>
<td>TokenGen</td>
<td>0.094</td>
<td>0.671</td>
</tr>
<tr>
<td>Search</td>
<td>0.961</td>
<td>0.529</td>
</tr>
</tbody>
</table>

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 \( \alpha = 28 \) and \( \beta = 10KB \) for Bloom filter so that \( m = \frac{10 \times 8 \times 1024}{45 \times 10^{-9}} \approx 40 \) and the false-positive rate is around \( 4.5 \times 10^{-9} \). 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 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 filters 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 primitive called verifiable attribute-based keyword search for secure cloud computing over outsourced encrypted data. This primitive allows a data owner to control the search of its outsourced encrypted data according to an access control policy, while the authorized data users can outsource the search operations to the cloud and force the cloud to faithfully execute the search (as a cheating cloud can be held accountable). Performance evaluation shows that the new primitive is practical. Our study focused on static data. As such, one interesting open problem for future research is to accommodate dynamic data.

Acknowledgement. Zheng and Xu were supported in part by the National Science Foundation under Grant No. 1111925. Atieniese was supported by a Google Faculty Research Award, an IBM Faculty Award, and the PRIN project TENECE.

REFERENCES

[38] “The java pairing based cryptography library. http://gas.dia.unisa.it/projects/jplbc/.”

APPENDIX A
PROOF OF THEOREM 1

Proof 1: We show that if there is a polynomial-time adversary A which wins the SCKA game with advantage µ, then there is a challenger algorithm that solves the DL problem with advantage µ/2. Given a DL instance \((g, h, f, r_1, r_2, Q, g^z)\), 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 pm = \((e, g, p, h, f^e, f)\) and mk = \((d)\).

A selects an attribute set \(\text{Atts'}\) and gives it to the challenger. The random oracle \(O_{H_2}(\text{Atts}')\) is defined as follows:
If $\alpha_j$ has not been queried before,
- if $\alpha_j \in \text{Atts}^*$, select $\beta_j \leftarrow \mathbb{Z}_p$, add $(\alpha_j, \beta_j)$ to $\mathcal{O}_{H_1}$, and return $g^{\beta_j}$;
- otherwise, select $\alpha_j, \beta_j \leftarrow \mathbb{Z}_p$, add $(\alpha_j, \alpha_j, \beta_j)$ to $\mathcal{O}_{H_1}$, and return $f^\alpha g^\beta$.

If $\alpha_j$ has been queried before, retrieve $(\alpha_j, \beta_j)$ from $\mathcal{O}_{H_1}$ and return $f^\alpha g^\beta$.

**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.

$\mathcal{O}_{\text{KeyGen}}(T)$: $\mathcal{A}$ gives an access tree $T$ to the challenger. If $F(\text{Atts}^*, T) = 1$, 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($T_v$, $\text{Atts}^*$, $\lambda_v$): Given secret $\lambda_v$, this procedure determines the polynomial for each node of $T_v$ rooted at $v$ when $F(\text{Atts}^*, T_v) = 1$. 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($T_{v'}$, $\text{Atts}^*$, $\lambda_{v'}$) where $\lambda_{v'} = q_v(\text{Index}(v'))$.

- PolyUnSat($T_v$, $\text{Atts}^*$, $g^{\lambda_v}$): Given element $g^{\lambda_v} \in \mathbb{G}$ where the secret $\lambda_v$ is unknown, this procedure determines the polynomial for each node of $T_v$ rooted at $v$ when $F(\text{Atts}^*, T_v) = 0$ 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 $F(\text{Atts}^*, T_{v'}) = 1$, then set $V = V \cup \{v'\}$. Because $F(\text{Atts}, T_v) = 0$, then $|V| < k_v$. For each node $v' \in V$, it selects $\lambda_{v'} \leftarrow \mathbb{Z}_p$, and sets $q_v(\text{Index}(v')) = \lambda_{v'}$. 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 $F(\text{Atts}^*, T_{v'}) = 1$, then run PolySat($T_{v'}$, $\text{Atts}^*$, $q_v(\text{Index}(v'))$, where $q_v(\text{Index}(v'))$ is known to the challenger;
- otherwise, call PolyUnSat($T_{v'}$, $\text{Atts}^*$, $g^{\lambda_{v'}}$), where $g^{\lambda_{v'}} = g^{q_v(\text{Index}(v'))}$ is known to the challenger.

With the above two procedures, the challenger runs $\mathcal{A}$ PolyUnSat($T$, $\text{Atts}^*$, $g^{\lambda}$), by implicitly defining $q_{\text{root}}(0) = \alpha$. Then for each $v \in \text{lv}(T)$, the challenger gets $g_v(0)$ if $\text{att}(v) \in \text{Atts}^*$, and gets $g^{q_v(0)}$ otherwise. Because $cq_v(0)$ is the secret share of $ac_v$ due to the linear property, the challenger generates credentials for each $v \in \text{lv}(T)$ as follows:

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

$$B_v = g^{q_v(0) - \frac{1}{\alpha_j}} g^t = g^{t - \frac{\alpha_j}{\alpha_j}}$$

$$A_v = g^{q_v(0) - \frac{\beta_j}{\alpha_j}} (f^\alpha g^t)^t$$

$$= f^{q_v(0)} (f^{\alpha_j} g^{\beta_j})^{\frac{\alpha_j}{\alpha_j}} (f^{\alpha} g^{t_j})^t$$

$$= f^{q_v(0)} (f^{\alpha_j} g^{\beta_j})^t = g^{q_v(0)} H_1(\text{att}(v))^t = g^{q_v(0)}$$

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 $\alpha_j, \beta_j$.

Eventually, the challenger returns $sk = \{ (A_v, B_v) \mid v \in \text{lv}(T) \}$ to $\mathcal{A}$.

$\mathcal{O}_{\text{TokenGen}}(T, w)$: The challenger runs $\mathcal{O}_{\text{KeyGen}}(T)$ to get $sk = \{ (A_v, B_v) \mid v \in \text{lv}(T) \}$, computes $tk \leftarrow \text{TokenGen}(sk, w)$, and returns $tk$ to $\mathcal{A}$. If $F(\text{Atts}, T) = 1$, 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 $\text{cph}^*$ as:

- Select $\lambda \leftarrow \{0, 1\}$.
- For each $\alpha_j \in \text{Atts}^*$, set $W_j = (g^{r_2})^{\beta_j}$.
- Set $W' = f^{r_1}, W = Q f^{r_1} d H_2(w_1), W_0 = g^{r_2}$, and for $\alpha_j \in \text{Atts}^*$, $W_j = (g^{r_2})^{\beta_j} = H_1(\alpha_j)^{r_2}$.

We note that if $Q = h^{r_1 + r_2}$, then $\text{cph}^*$ is indeed a legitimate ciphertext for keyword $w_\lambda$. The reason is that $W' = f^{r_1} = g^{r_1}, W = Q f^{r_1} d H_2(w_1) = Q g^{r_1 c d H_2(w_1)} = g^{\alpha_j r_1 + r_2} g^{\beta_j H_2(w_1)}$, $W_0 = g^{r_2}$, and for $\alpha_j \in \text{Atts}^*$, $W_j = (g^{r_2})^{\beta_j} = H_1(\alpha_j)^{r_2}$.

**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 $\mathcal{O}_{\text{TokenGen}}$ if $F(\text{Atts}, T) = 1$.

**Guess:** Finally, $\mathcal{A}$ outputs a bit $\lambda'$ 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 $\text{cph}^*$ is a valid ciphertext of $w_\lambda$, so the probability of $\mathcal{A}$ outputting $\lambda' = \lambda'$ is $\frac{1}{2} + \mu$. If $Q$ is an element randomly selected from $G$, then $\text{cph}^*$ is not a valid ciphertext of $w_\lambda$. 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 $Q \triangleq h^{r_1 + r_2}$ with the DL instance $(g, h, f, f^{\tau_1}, g^{r_2}, Q)$ 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 $\mu$.

**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 $pr = (e, g, g^a, g^b, g^c, f)$ and $mk = (a, b, c)$. 
The random oracle $O_{H_1}(a_{t_j})$ is simulated as follows: If $a_{t_j}$ has not been queried before, the challenger selects $\alpha_j \leftarrow \mathbb{Z}_p$, adds $(a_{t_j}, \alpha_j)$ to $O_{H_1}$, and returns $g^{\alpha_j}$; otherwise, the challenger retrieves $\alpha_j$ from $O_{H_1}$ and returns $g^{\alpha_j}$.

**Phase 1:** A can adaptively query the following oracles for polynomially-many times.

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

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

**Challenge Phase:** $A$ selects an attribute set $Atts^*$. The challenger chooses an access control policy that is represented as $T^*$ such that $F(Atts^*, T^*) = 1$, computes $sk^* \leftarrow \text{KeyGen}(mk, T^*)$. 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, $A$ outputs a keyword $w^*$ and gives it to the challenger. The challenger computes $cph^* \leftarrow \text{Enc}(Atts^*, w^*)$ and if $\text{Search}(tk^*, cph^*) = 1$, then $A$ wins the game.

This finishes the simulation. Suppose $A$ has already attempted $q$ distinct keywords before outputting $w^*$, we can see that the probability of $A$ winning the keyword secrecy game is at most $\frac{1}{\sqrt{|M|}} + \epsilon$. This is because the size of the remaining keyword space is $|M| - q$, 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 $\epsilon$. Therefore, given $q$ distinct keywords $A$ has attempted, the probability of $A$ winning the keyword secrecy game is at most $\frac{1}{\sqrt{|M|}} + \epsilon$. Thus, our scheme achieves keyword secrecy as in Definition 3.

**Appendix C**

**Proof of Theorem 3 on CP-ABKS**

**Proof 3:** 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, $A$ attempts to distinguish $g^{\alpha_j(a_{t_j}+r_j)} h^{\alpha_j H_2(u_j)}$ from $g^{\alpha_j(a_{t_j}+r_j)} h^{\alpha_j H_2(u_j)}$. Given $\theta \leftarrow \mathbb{Z}_p$, the probability of distinguishing $g^{\alpha_j(a_{t_j}+r_j)} h^{\alpha_j H_2(u_j)}$ from $g^{\alpha_j(a_{t_j}+r_j)} h^{\alpha_j H_2(u_j)}$ is equal to that of distinguishing $g^{\theta}$ from $g^{\theta} h^{\alpha_j H_2(u_j)}$. Therefore, if $A$ has advantage $\epsilon$ in breaking the SCKA game, then it has advantage $\epsilon/2$ in distinguishing $g^{\alpha_j(a_{t_j}+r_j)} h^{\alpha_j H_2(u_j)}$ from $g^{\alpha_j(a_{t_j}+r_j)}$. Thus, let us consider a modified game where $A$ can distinguish $g^{\alpha_j(a_{t_j}+r_j)}$ from $g^{\theta}$. 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 $A$. $A$ chooses an access tree $T^*$, which is sent to the challenger.

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

**Phase 1:** $A$ can query $O_{\text{KeyGen}}$ and $O_{\text{TokenGen}}$ as follows:

**Phase 2:** The challenger selects $r_{(t)} \leftarrow \mathbb{Z}_p$ and computes $A = g^{(ac+r_{(t)})/b}$. For each attribute $a_{t_j} \in Atts$, the challenger chooses $r_{(t)_j} \leftarrow \mathbb{Z}_p$, computes $A_j = g^{(ac+r_{(t)_j})/b}$ and $B_j = g^{a_{t_j}}$, and returns $(Atts, A, \{(A_j, B_j) | a_{t_j} \in Atts\})$.

**Phase 3:** $A$ can query $O_{\text{KeyGen}}$, $O_{\text{TokenGen}}$ and $O_{\text{Search}}$ as follows:

<table>
<thead>
<tr>
<th>$a$</th>
<th>$cph^*$</th>
<th>$tk^*$</th>
<th>$ws$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{(t)}$</td>
<td>$cph^*(r_{(t)})$</td>
<td>$tk^*$</td>
<td>$ws$</td>
</tr>
<tr>
<td>$ac+r_{(t)}$</td>
<td>$cph^*(ac+r_{(t)})$</td>
<td>$tk^*$</td>
<td>$ws$</td>
</tr>
</tbody>
</table>

**Possible Terms for Querying Group Oracle $G_T$**
only can be reconstructed if the attributes corresponding to \( r_j^{(t)} \) of \( r_j^{(t)} + \alpha_j r_j^{(t)} \) 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 4:** We show that if there exists a polynomial-time algorithm \( \mathcal{A} \) breaks VABKS’s data secrecy with the advantage \( \rho \), then we can break either CPA security for ABE or CPA security for SE with the advantage \( \frac{\rho}{n} \) 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 \( D_0 = (KG, MP, \{F_{01}, \ldots, F_{0n}\}) \), \( D_1 = (KG, MP, \{F_{11}, \ldots, F_{1n}\}) \), \( \{\lambda_{\text{Enc}}\}_1 \) and \( \{\lambda'_{\text{Enc}}\}_n \). The challenge selects \( \lambda \leftarrow \{0, 1\} \) and encrypts \( FS_{\lambda} \) with the ABE and \( \{\lambda_{\text{Enc}}\}_n \).

Now let us consider the advantage of \( \mathcal{A} \) correctly guessing \( \lambda \). 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 \( \rho \), then the advantage of distinguishing which data file was encrypted is \( \frac{\rho}{n} \) 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 with a non-negligible advantage \( \rho \), then the advantage of breaking CPA security for ABE or CPA security for SE is \( \frac{\rho}{n} \) -- 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 5:** 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 \( \rho \), then we can break the selective security against chosen-keyword attack game of ABKS with the advantage of \( \frac{\rho}{n} \), 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 \( D_0 = (KG_0, \{KG_{01}, \ldots, KG_{0n}\}, MP, FS) \), and \( \{\lambda'_{\text{Enc}}\}_n \). The challenge selects \( \lambda \leftarrow \{0, 1\} \) and encrypts KG with ABKS, and generates \( BF_i, \text{cph}_{BF_i} \), and \( \sigma_i \) for each keyword group.

Since ABE is CPA-secure and \( H \) is a secure pseudorandom generator, the probability of \( \mathcal{A} \) inferring \( \lambda \) via \( BF_i, \text{cph}_{BF_i} \), is negligible. Then let us consider the advantage of \( \mathcal{A} \) correctly guessing \( \lambda \) 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 \( KG_0 \) and \( KG_1 \), if the advantage of distinguishing which keyword set was encrypted is \( \rho \), then the advantage of distinguishing which keyword was encrypted is \( \frac{\rho}{n} \) by selecting one keyword from \( KG_0 \) and one from \( KG_1 \).

Therefore, we can see that if \( \mathcal{A} \) breaks VABKS’s selective security against chosen-keyword attack with a non-negligible advantage \( \rho \), then the advantage of breaking ABKS’s selective security against chosen-keyword attack is \( \frac{\rho}{n} \) -- 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 6:** We show that if there exists polynomial time algorithm \( \mathcal{A} \) breaking VABKS’s keyword secrecy, then it breaks the assumption that ABKS achieves keyword secrecy.

Suppose \( \mathcal{A} \) presents a data collection \( D = (KG = \{KG_0, \ldots, KG_l\}, MP, FS), \{I_{\text{Enc}}\}_1 \) and \( \{I'_{\text{Enc}}\}_l \). 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 \( \frac{1}{|M|} + \epsilon \) after guessing \( q \) distinct keywords, then the probability of guessing the keyword from the search token and keyword ciphertext in ABKS is more than \( \frac{1}{|M|} + \epsilon \) after guessing \( q \) distinct keywords, which contracts the assumption that ABKS achieves keyword secrecy.

**D. Proof of Theorem 8 on VABKS**

**Proof 7:** 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 \( D = (KG = \{KG_0, \ldots, KG_l\}, MP, FS), \{I_{\text{Enc}}\}_1 \) and \( \{I'_{\text{Enc}}\}_l \). The challenger runs \((Au, \text{Index}, D_{\text{cph}}) \leftarrow \text{BuildIndex}(\{I_{\text{Enc}}\}_1, \{I'_{\text{Enc}}\}_l, D)\), and gives \((Au, \text{Index}, D_{\text{cph}})\) to \( \mathcal{A} \).

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

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

First, we claim that the global signature \( \sigma \) and random keyword ciphertexts \( \text{cph}_{BF_1}, \ldots, \text{cph}_{BF} \) 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 $\text{VABKS.Verify}$ output 1. The reason is that $\mathcal{A}$ cannot forge a keyword signature $\sigma_w$ for the keyword ciphertext and data ciphertexts; otherwise, we can break the unforgeability of $\text{Sig}$. 

- 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 $\text{VABKS.Verify}$ output 1. Suppose $\mathcal{A}$ returns a null result and the proof $(\text{BF}'_i, \text{cph}_{\text{BF}'_i}, \sigma_{\text{BF}'_i})$. Since $\text{BF}'_i$ cannot be manipulated due to $\sigma_{\text{BF}'_i}$, the unmasked bloom filter indicates that $w^*$ is a member within the group. The challenger downloads $\text{cph}_{w_1}, \ldots, \text{cph}_{w_1|KG_i}$ and $\sigma_i$ without being manipulated; otherwise we break the $\text{Sig}$’s unforgeability. Then the challenger can conduct the search operation with each keyword ciphertext, and $\text{VABKS.Verify}$ will output 0. That is, if there exists keyword ciphertext matched the search token, $\mathcal{A}$ returns a null result, then it cannot make $\text{VABKS.Verify}$ output 1.

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