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.

978-1-4799-3360-0/14/$31.00 ©2014 IEEE

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

Authorized licensed use limited to: Indian Institute of Technology - BHUBANESWAR. Downloaded on October 13,2024 at 05:50:22 UTC from IEEE Xplore. Restrictions apply.

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 $\mathsf{KP\text{-}ABKS}$ and $\mathsf{CP\text{-}ABKS}$ 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 $\mathsf{string}(S)$ denote the concatenation of elements of S ordered by their hash values. Let $\mathsf{U} = \{\mathsf{at}_1, \ldots, \mathsf{at}_n\}$ be a set of attributes that are used to specify access control policies.

A. Cryptographic Assumption

Let p be an ℓ-bit prime, and G, G_T be cyclic groups of prime order p with generators g, g_T, respectively. Let e be a bilinear map: $e : G \times G \to G_T$ 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 $Q \stackrel{?}{=} h^{r_1+r_2}$ at most with a negligible advantage in security parameter ℓ, where "advantage" is defined as

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

B. Bloom Filter for Membership Query

A Bloom filter [33] 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 $(1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-kn/m})^k$. 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:

- $\mathsf{BF} \leftarrow \mathsf{BFGen}(\{H'_1, \ldots, H'_k\}, \{w_1, \ldots, w_n\})$: This algorithm generates a m-bit Bloom filter by hashing a data set $S = \{w_1, \ldots, w_n\}$ with $\{H'_1, \ldots, H'_k\}$.
- $\{0, 1\} \leftarrow \mathsf{BFVerify}(\{H'_1, \ldots, H'_k\}, \mathsf{BF}, w)$: This algorithm returns 1 if $w \in S$, 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 num_v be the number of children of node v, and label the children from the left to the right as $1, \ldots, \mathsf{num}_v$. Let k_v, $1 \leq k_v \leq \mathsf{num}_v$, be the threshold value associated with node v, where $k_v = 1$ represents the OR gate and $k_v = num_v$ represents the AND gate. Let $\mathsf{parent}(v)$ denote the parent of node v, $\mathsf{ind}(v)$ denote the label of node v, $\mathsf{att}(v)$ denote the attribute associated to leaf node v, $\mathsf{lvs}(\mathsf{T})$ denote the set of leaves of access tree T, and T_v denote the subtree of T rooted at node v (e.g., $\mathsf{T}_{\mathsf{root}} = \mathsf{T}$).

Let $F(\mathsf{Atts}, \mathsf{T}_v) = 1$ indicate that an attribute set Atts satisfies the access control policy represented by subtree T_v, where $F(\mathsf{Atts}, \mathsf{T}_v)$ can be evaluated iteratively as follows:

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

Authorized licensed use limited to: Indian Institute of Technology - BHUBANESWAR. Downloaded on October 13,2024 at 05:50:22 UTC from IEEE Xplore. Restrictions apply.

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

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

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., $v = \mathsf{root}$), set $q_v(0) = s$ and randomly pick $k_v - 1$ coefficients for polynomial q_v.
- If v is a leaf of T, set $q_v(0) = q_{\mathsf{parent}(v)}(\mathsf{ind}(v))$.
- If v is an inner node (but not the root), 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 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, $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}) = 1$, $E_{u_j} = e(g, h)^{q_{u_j}(0)}$ 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, the algorithm for reconstructing $e(g, h)^s$ is denoted by

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

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

- If $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_v) = 0$, then set $E_v = \perp$.
- If $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_v) = 1$, 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 $\{v_1, \cdots, v_{\mathsf{num}_v}\}$, there exists a set of indices S such that $|S| = k_v$, $j \in S$, and $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_{v_j}) = 1$. Set
    $$E_v = \prod_{j \in S} E_{v_j}^{\Delta_{v_j}} = \prod_{j \in S} (e(g, h)^{q_{v_j}(0)})^{\Delta_{v_j}} = e(g, h)^{q_v(0)},$$
    where $\Delta_{v_j} = \prod_{l \in S, l \neq j} \frac{-j}{l - j}$.

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

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 outsoured encrypted data. That is, a data user who possesses attributes that satisfy the data owner's policy can conduct keyword search over the oursourced 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_{Enc} denote the input to encryption function Enc and I_{KeyGen} denote the input to key generation function KeyGen. For CP-ABKS, I_{Enc} and I_{KeyGen} are respectively the access tree and the attribute set; for KP-ABKS, I_{Enc} and I_{KeyGen} are respectively the attribute set and the access tree. Let $F(I_{\mathsf{KeyGen}}, I_{\mathsf{Enc}}) = 1$ denote I_{KeyGen} satisfies I_{Enc} in CP-ABKS and I_{Enc} satisfies I_{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:

- $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Setup}(1^\ell)$: This algorithm initializes the public parameter pm and generates a master key mk.
- $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: This algorithm outputs credential sk for a user according to I_{KeyGen}.
- $\mathsf{cph} \leftarrow \mathsf{Enc}(w, I_{\mathsf{Enc}})$: This algorithm encrypts keyword w to obtain ciphertext cph.
- $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$: This algorithm allows a data user to generate a search token tk according to its credential sk and keyword w.
- $\{0, 1\} \leftarrow \mathsf{Search}(\mathsf{cph}, \mathsf{tk})$: This algorithm returns 1 if (i) $F(I_{\mathsf{KeyGen}}, I_{\mathsf{Enc}}) = 1$ 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 $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Setup}(1^\ell)$, $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$ and $F(I_{\mathsf{KeyGen}}, I_{\mathsf{Enc}}) = 1$, $\mathsf{cph} \leftarrow \mathsf{Enc}(w, I_{\mathsf{Enc}})$ and $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$, $\mathsf{Search}(\mathsf{cph}, \mathsf{tk})$ 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 (formal definition can be found in the full version of the present paper [34]):

- *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_{Enc} it intends to attack before the system is boostrapped [35].
- *Keyword secrecy*: In the public-key setting, it is impossible to protect the search tokens (aka. predicate privacy [36]) 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 $\mathcal{A}$ learning the keyword from the keyword ciphertext and search tokens is negligibly more than the probability of correct random keyword guess.

Authorized licensed use limited to: Indian Institute of Technology - BHUBANESWAR. Downloaded on October 13,2024 at 05:50:22 UTC from IEEE Xplore. Restrictions apply.

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 $t \leftarrow \mathbb{Z}_p$, $A_v = g^{q_v(0)} H_1(\mathsf{att}(v))^t, B_v = g^t$ 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 $W' = g^{cr_1}$, $W = g^{a(r_1+r_2)} g^{bH_2(w)r_1}$ and $W_0 = g^{r_2}$ where $g^a, g^b, g^c \in G$ are public keys, and the other is associated to 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 together via r_2.
- Given a set of credentials, a search token for keyword w is generated with two parts: one is associated to w as $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s$ and $\mathsf{tok}_2 = g^{cs}$ for some $s \leftarrow \mathbb{Z}_p$, 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 together via randomness s.

If the attribute set Atts 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.

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

$\mathsf{Setup}(1^\ell)$: 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 ℓ-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

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

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

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

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

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

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

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

$\mathsf{Search}(\mathsf{tk}, \mathsf{cph})$: Given attribute set Atts specified in cph, select an attribute set S satisfying the access tree T specified in tk. If S does not exist, return 0; otherwise, for each $\mathsf{at}_j \in S$, compute $E_v = e(A'_v, W_0)/e(B'_v, W_j) = e(g,g)^{sr_2 q_v(0)}$, where $\mathsf{att}(v) = \mathsf{at}_j$ for $v \in \mathsf{lvs}(\mathsf{T})$. Compute $e(g,g)^{sr_2 q_{\mathsf{root}}(0)} \leftarrow \mathsf{Combine}(\mathsf{T}, \{E_v | \mathsf{att}(v) \in S\})$ so that $E_{\mathsf{root}} = e(g,g)^{acsr_2}$. Return 1 if $e(W', \mathsf{tok}_1) E_{\mathsf{root}} = e(W, \mathsf{tok}_2)$, and 0 otherwise.

The scheme is correct because

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

The scheme is secure because of the following theorems, whose proofs are given in the full version of the paper [34].

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 ℓ be the primary security parameter. It consists of the following algorithms.

$\mathsf{Setup}(1^\ell)$: 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 ℓ-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

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

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

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

$\mathsf{Enc}(w, \mathsf{T})$: Select $r_1, r_2 \leftarrow \mathbb{Z}_p$, and compute $W = g^{cr_1}$, $W_0 = g^{a(r_1+r_2)} g^{bH_2(w)r_1}$ and $W' = g^{br_2}$. Compute secret shares of r_2 for each leave of access tree T as $\{q_v(0) | v \in \mathsf{lvs}(\mathsf{T})\} \leftarrow \mathsf{Share}(\mathsf{T}, r_2)$. For each $v \in \mathsf{lvs}(\mathsf{T})$, compute $W_v = g^{q_v(0)}$ and $D_v = H_1(\mathsf{att}(v))^{q_v(0)}$. Set

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

$\mathsf{TokenGen}(\mathsf{sk}, w)$: Select $s \leftarrow \mathbb{Z}_p$, and compute $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s, \mathsf{tok}_2 = g^{cs}$ and $\mathsf{tok}_3 = A^s = g^{(acs-rs)/b}$. For each $\mathsf{at}_j \in \mathsf{Atts}$, compute $A'_j = A_j^s$ and $B'_j = B_j^s$. Set

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

$\mathsf{Search}(\mathsf{tk}, \mathsf{cph})$: Given attribute set Atts as specified in tk, select an attribute set S that satisfies the access tree T specified in cph. If S does not exist, return 0; otherwise, for each $\mathsf{at}_j \in S$, compute $E_v = e(A'_j, W_v)/e(B'_j, D_v) =$

Authorized licensed use limited to: Indian Institute of Technology - BHUBANESWAR. Downloaded on October 13,2024 at 05:50:22 UTC from IEEE Xplore. Restrictions apply.

$e(g,g)^{rsq_v(0)}$, where $\mathsf{att}(v) = \mathsf{at}_j$ for $v \in \mathsf{lvs}(\mathsf{T})$. Compute $e(g,g)^{rsq_{\mathsf{root}}(0)} \leftarrow \mathsf{Combine}(\mathsf{T}, \{E_v | \mathsf{att}(v) \in S\})$ and $E_{\mathsf{root}} = e(g,g)^{rsr_2}$. Return 1 if $e(W_0, \mathsf{tok}_2) = e(W, \mathsf{tok}_1) E_{\mathsf{root}} e(\mathsf{tok}_3, W')$, 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, whose proofs are given in the full version [34].

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 [37].

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).

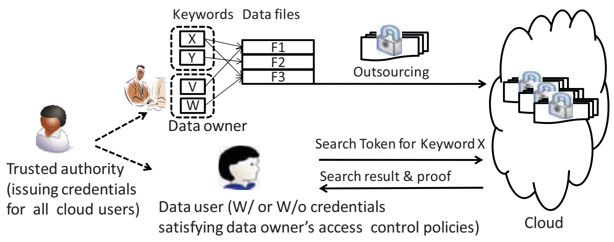

Keywords
Data files
X
Y
V
W
F1
F2
F3
Outsourcing
Data owner
Trusted authority (issuing credentials for all cloud users)
Search Token for Keyword X
Search result & proof
Data user (W/ or W/o credentials satisfying data owner's access control policies)
Cloud

Fig. 1. VABKS system model, where keywords X, Y and V, W may correspond to different access control policies.

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 $\mathsf{FS} = \{\mathsf{F}_1, \dots, \mathsf{F}_n\}$ be a set of data files. Let KG_j, $1 \le j \le l$, be a set of keywords (also called "keyword group") that are encrypted with the same access control policy (i.e., access tree). Let $\mathsf{KG} = \{\mathsf{KG}_1, \dots, \mathsf{KG}_l\}$. For each keyword w, let $\mathsf{MP}(w)$ be the set of identifiers identifying data files that contain keyword w. Let $\mathsf{MP} = \{\mathsf{MP}(w) | w \in \cup_{i=1}^{l} \mathsf{KG}_i\}$. Let $\mathsf{D} = (\mathsf{KG}, \mathsf{MP}, \mathsf{FS})$ denote keyword-index and the data files.

Definition 2: A VABKS scheme consists of the following algorithms:

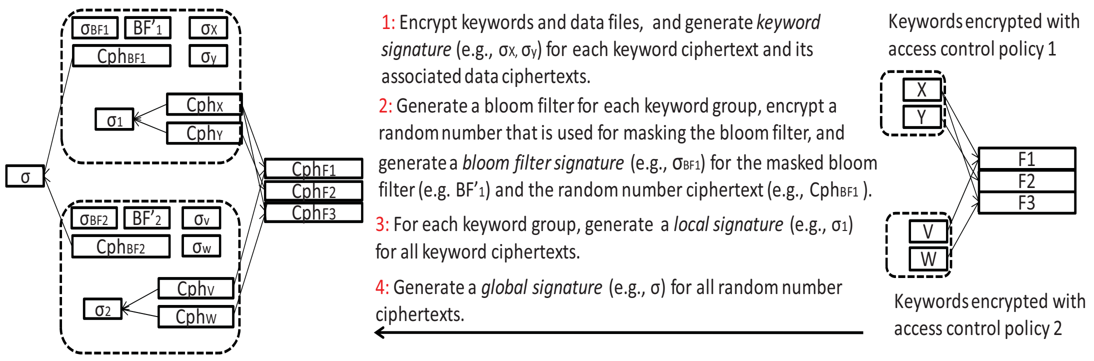
- $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Init}(1^\ell)$: This algorithm is run by the trusted authority to initialize the system.
- $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: This algorithm is run by the trusted authority to issue credentials sk for data users/owners.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

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

Informally, security of VABKS is defined as the following four requirements, where the cloud is the adversary $\mathcal{A}$ (formal definitions are given in the full version [34]).

- *Data secrecy*: Given encrypted keywords and search tokens, $\mathcal{A}$ 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 $\mathsf{D}_0 = (\mathsf{KG}, \mathsf{MP}, \mathsf{FS}_0)$, $\mathsf{D}_1 = (\mathsf{KG}, \mathsf{MP}, \mathsf{FS}_1)$ correspond to the same KG and MP, and $|\mathsf{FS}_0| = |\mathsf{FS}_1|$.
- *Selective security against chosen-keyword attack*: Without seeing corresponding search tokens, $\mathcal{A}$ 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 $\mathcal{A}$ learn 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.

Authorized licensed use limited to: Indian Institute of Technology - BHUBANESWAR. Downloaded on October 13,2024 at 05:50:22 UTC from IEEE Xplore. Restrictions apply.


σBF1
BF'1
σX
σY
CphBF1
σ1
CphX
CphY
σ
σBF2
BF'2
σV
σW
CphBF2
σ2
CphV
CphW
CphF1
CphF2
CphF3
Keywords encrypted with access control policy 1
X
Y
F1
F2
F3
V
W
Keywords encrypted with access control policy 2

1: Encrypt keywords and data files, and generate *keyword signature* (e.g., σX, σY) for each keyword ciphertext and its associated data ciphertexts.

2: Generate a bloom filter for each keyword group, encrypt a random number that is used for masking the bloom filter, and generate a *bloom filter signature* (e.g., σBF1) for the masked bloom filter (e.g. BF'1) and the random number ciphertext (e.g., CphBF1).

3: For each keyword group, generate a *local signature* (e.g., σ1) for all keyword ciphertexts.

4: Generate a *global signature* (e.g., σ) for all random number ciphertexts.

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 keyword group i, the cloud provides $(\sigma_w, \mathsf{cph}_{\mathsf{BF}_i})$ as the proof when it finds keyword ciphertext cph_w that matches tk, and $(\mathsf{cph}_{\mathsf{BF}_i}, \mathsf{BF}'_i, \sigma_{\mathsf{BF}_i})$ otherwise.

- *Verifiability*: If $\mathcal{A}$ returns an incorrect search result, it can be detected by the user with an overwhelming probability.

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 KG_j. 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 $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, a symmetric encryption scheme $\mathsf{SE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, an ABE scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where the latter two encryption schemes are used to encrypt data files. The VABKS scheme is built on top of an ABKS scheme $\mathsf{ABKS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{TokenGen}, \mathsf{Search})$, 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 $\{\mathsf{cph}_w | w \in \mathsf{KG}_i\}$, where the user searches keyword w', it can happen that $1 \leftarrow \mathsf{BFVerify}(\{H'_1, \ldots, H'_k\}, \mathsf{BF}_i, w')$ due to the false-positive of the Bloom filter. To validate the search result in this case, the Verify algorithm has to download $\{\mathsf{cph}_w | w \in \mathsf{KG}_i\}$, 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 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 the full version [34].

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,

Authorized licensed use limited to: Indian Institute of Technology - BHUBANESWAR. Downloaded on October 13,2024 at 05:50:22 UTC from IEEE Xplore. Restrictions apply.

$\underline{\mathsf{Init}}(1^\ell)$: Given security parameter ℓ, the attribute authority chooses k universal hash functions $H'_1,\ldots,H'_k$, which are used to construct 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 $(\mathsf{ABE.pm},\mathsf{ABE.mk}) \leftarrow \mathsf{ABE.Setup}(1^\ell)$ and $(\mathsf{ABKS.pm},\mathsf{ABKS.mk}) \leftarrow \mathsf{ABKS.Setup}(1^\ell)$. It sets the public parameter as $\mathsf{pm} = (\mathsf{ABE.pm},\mathsf{ABKS.pm},H'_1,\ldots,H'_k)$ 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 a 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 $\mathsf{D} = (\mathsf{KG} = \{\mathsf{KG}_1,\ldots,\mathsf{KG}_l\}, \mathsf{MP} = \{\mathsf{MP}(w) | w \in \cup_{i=1}^l \mathsf{KG}_i\}, \mathsf{FS} = \{\mathsf{F}_1,\ldots,\mathsf{F}_n\})$, the data owner executes as follows:

1) Encrypt each data file with hybrid encryption: $\forall \mathsf{F}_j \in \mathsf{FS}$, generate 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 $\mathsf{KG}_i, 1 \le i \le l$, for each $w \in \mathsf{KG}_i$, run $\mathsf{cph}_w \leftarrow \mathsf{ABKS.Enc}(I_{\mathsf{Enc}_i}, w)$, set $\mathsf{MP}(\mathsf{cph}_w) = \{\mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} | \mathsf{ID}_{\mathsf{F}_j} \in \mathsf{MP}(w)\}$, and generate $\sigma_w \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{cph}_w || \mathsf{string}(\{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\}))$, where $\mathsf{ID}_{\mathsf{F}_j}$ and $\mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}}$ are identifiers 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 KG_i: Let $\mathsf{BF}_i \leftarrow \mathsf{BFGen}(\{H'_1,\ldots,H'_k\}, \mathsf{KG}_i)$, $\mathsf{cph}_{\mathsf{BF}_i} \leftarrow \mathsf{ABE.Enc}(I_{\mathsf{Enc}_i}, \mathsf{M})$ for some randomly chosen 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 $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{string}(\{\mathsf{cph}_w | w \in \mathsf{KG}_i\}))$.
4) Generate the global signature: Set $\sigma = \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{cph}_{\mathsf{BF}_1} || \ldots || \mathsf{cph}_{\mathsf{BF}_l})$.
5) Let $\mathsf{Au} = (\sigma, \sigma_1, \ldots, \sigma_l, \mathsf{cph}_{\mathsf{BF}_1}, \ldots, \mathsf{cph}_{\mathsf{BF}_l}, \sigma_{\mathsf{BF}_1}, \ldots, \sigma_{\mathsf{BF}_l}, \{\sigma_w | w \in \cup_{i=1}^l \mathsf{KG}_i\})$, $\mathsf{Index} = (\{\mathsf{cph}_w | w \in \cup_{i=1}^l \mathsf{KG}_i\}, \{\mathsf{MP}(\mathsf{cph}_w) | w \in \cup_{i=1}^l \mathsf{KG}_i\})$ 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 $\prod_i = \{\mathsf{cph}_w | w \in \mathsf{KG}_i\}, 1 \le i \le l$, 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 $\gamma = 0$, it continues to process the 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 processing all cph_w in $\prod_i$, then its adds $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i})$ 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 the integrity of the random number ciphertexts: Let $\gamma = \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma, \mathsf{cph}_{\mathsf{BF}_1} || \ldots || \mathsf{cph}_{\mathsf{BF}_l})$. 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 the correct result for each keyword group i:
   <u>Case 1:</u> If $(\mathsf{cph}_w, \{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\}) \in \mathsf{rslt}$, meaning there exists the keyword ciphertext cph_w, which corresponds to the same access control policy as what is specified by $\mathsf{cph}_{\mathsf{BF}_i}$, having the same keyword specified by tk, 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, continue to $i = i + 1$.
   <u>Case 2:</u> If $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i}) \in \mathsf{proof}$ meaning that there is no matching keyword ciphertext, then it continues to verify the integrity of the masked Bloom filter by running $\gamma' \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_{\mathsf{BF}_i}, \mathsf{BF}'_i || \mathsf{cph}_{\mathsf{BF}_i})$. If $\gamma' = 0$, return 0; otherwise, execute the following:
   - If the data user is authorized, compute $\mathsf{M} \leftarrow \mathsf{ABE.Dec}(\mathsf{ABE.sk}, \mathsf{cph}_{\mathsf{BF}_i})$, $\mathsf{BF}_i = H(\mathsf{M}) \bigotimes \mathsf{BF}'_i$. Execute $\delta \leftarrow \mathsf{BFVerify}(\{H'_1,\ldots,H'_k\}, \mathsf{BF}_i, w)$ 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 $\prod_i = \{\mathsf{cph}_w | w \in \mathsf{KG}_i\}$ and σ_i from the cloud, and run $\eta \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_i, \mathsf{string}(\{\mathsf{cph}_w | w \in \mathsf{KG}_i\}))$. If $\eta = 0$, return 0; otherwise, run $\tau \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$ by enumerating cph_w in $\mathsf{cph}_w | w \in \mathsf{KG}_i\}$. 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 $\mathsf{cph}_{\mathsf{BF}_i}$ cannot be decrypted.

   <u>Case 3:</u> 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.

Fig. 3. VABKS construction

Authorized licensed use limited to: Indian Institute of Technology - BHUBANESWAR. Downloaded on October 13,2024 at 05:50:22 UTC from IEEE Xplore. Restrictions apply.

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 instantiated with any secure solutions. 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, while using the Java Pairing Based Cryptography library (jPBC) [38]. 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-VABKS and KP-VABKS, 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-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.

		complexity	output size
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	$(2S+2)\mathsf{Pair} + S\mathsf{E}_T$	
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).

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 II shows the execution time of the two ABKS schemes. 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 that the data users outsource the keyword search operations to the cloud (i.e., taking advantage of the cloud's computational resources).

		S/N					
		1	10	20	30	40	50
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
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.

B. *Efficiency of* VABKS *with Real Data*

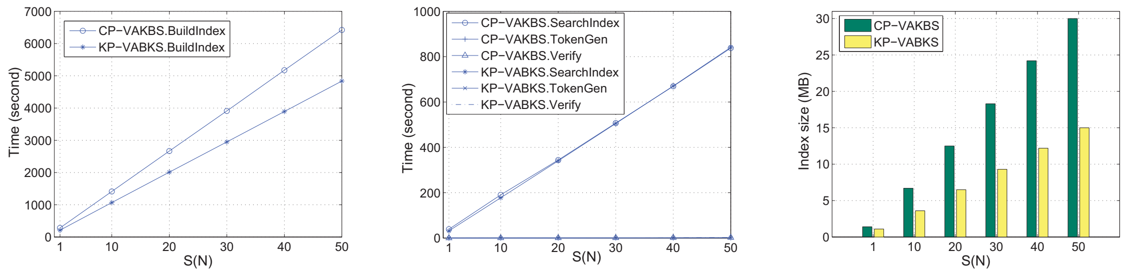
To demonstrate the feasibility of VABKS in practice, we evaluated it with real data, which consists of 2,019 distinct keywords extracted from 670 PDF documents (papers) from the ACM Digital Library with a total size of 778.1MB. We set $k = 28$ and $m = 10KB$ for Bloom filter so that $\frac{m}{n} = \frac{10*8*1024}{2019} \approx 40$ 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 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

Authorized licensed use limited to: Indian Institute of Technology - BHUBANESWAR. Downloaded on October 13,2024 at 05:50:22 UTC from IEEE Xplore. Restrictions apply.



(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.

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. Ateniese was supported by a Google Faculty Research Award, an IBM Faculty Award, and the PRIN project TENACE.

REFERENCES

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

[2] 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.

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

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

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

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

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

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

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

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

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

[16] 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.

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

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

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

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

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

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

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

[24] 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.

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

[26] 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.

[27] 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.

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

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

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

[34] Q. Zheng, S. Xu, and G. Ateniese, "Vabks: Verifiable attribute-based keyword search over outsourced encrypted data." Cryptology ePrint Archive, Report 2013/462, 2013. http://eprint.iacr.org.

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

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

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

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

Authorized licensed use limited to: Indian Institute of Technology - BHUBANESWAR. Downloaded on October 13,2024 at 05:50:22 UTC from IEEE Xplore. Restrictions apply.