INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES
&
MANAGEMENT
Ijesm
Consulting, help, relaxation

Hierarchical Attribute-Based Solution for Flexible and Scalable Access Control in Cloud Computing

*k.venkateswarlu (M.tech pursuing), P.B.R.vits.Kavali,venkateswarlu.venkey9@gmail.com
**K.Venkata subbaiah,Assoc.proff, P.B.R.vits.Kavali,venkateswarlu.venkey9@gmail.com

***Abstract*—Cloud computing has emerged as one of the most influential paradigms in the IT industry in recent years. Since this new computing technology requires users to entrust their valuable data to cloud providers, there have been increasing security and privacy concerns on outsourced data. Several schemes employing attribute-based encryption (ABE) have been proposed for access control of outsourced data in cloud computing; however, most of them suffer from inflexibility in implementing complex access control policies. In order to realize scalable, flexible, and fine-grained access control of outsourced data in cloud computing, in this paper, we propose hierarchical attribute-set-based encryption (HASBE) by extending ciphertext-policy attribute-set-based encryption (ASBE) with a hierarchical structure of users. The proposed scheme not only achieves scalability due to its hierarchical structure, but also inherits flexibility and fine-grained access control in supporting compound attributes of ASBE. In addition, HASBE employs multiple value assignments for access expiration time to deal with user revocation more efficiently than existing schemes. We formally prove the security of HASBE based on security of the ciphertext-policy attribute-based encryption (CP-ABE) scheme by Bethencourt *et al.* and analyze its performance and computational complexity. We implement our scheme and show that it is both efficient and flexible in dealing with access control for outsourced data in cloud computing with comprehensive experiments.**

***Index Terms*—Access control, cloud computing, data security.**

I. Introduction

CLOUD computing is a new computing paradigm that is built on virtualization, parallel and distributed computing, utility computing, and service-oriented architecture. In the last several years, cloud computing has emerged as one of the most influential paradigms in the IT industry, and has attracted extensive attention from both academia and industry. Cloud computing holds the promise of providing computing as the fifth utility [1] after the other four utilities (water, gas, electricity, and telephone). The benefits of cloud computing include reduced costs and capital expenditures, increased operational efficiencies, scalability, flexibility, immediate time to market, and so on. Different service-oriented cloud computing models have been proposed, including Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Numerous commercial cloud computing systems have been built at different levels, e.g., Amazon's EC2 [2], Amazon's S3 [3], and IBM's Blue Cloud [4] are IaaS systems, while Google App Engine [5] and Yahoo Pig are representative PaaS systems, and Google's Apps [6] and Salesforce's Customer Relation Management (CRM) System [7] belong to SaaS systems. With these cloud computing systems, on one hand, enterprise users no longer need to invest in hardware/software systems or hire IT professionals to maintain these IT systems, thus they save cost on IT infrastructure and human resources; on the other hand, computing utilities provided by cloud computing are being offered at a relatively low price in a pay-as-you-use style. For example, Amazon's S3 data storage service with 99.99% durability charges only $0.06 to $0.15 per gigabyte-month, while traditional storage cost ranges from $1.00 to $3.50 per gigabyte-month according to Zetta Inc. [8].

Although the great benefits brought by cloud computing paradigm are exciting for IT companies, academic researchers, and potential cloud users, security problems in cloud computing become serious obstacles which, without being appropriately addressed, will prevent cloud computing's extensive applications and usage in the future. One of the prominent security concerns is data security and privacy in cloud computing due to its Internet-based data storage and management. In cloud computing, users have to give up their data to the cloud service provider for storage and business operations, while the cloud service provider is usually a commercial enterprise which cannot be totally trusted. Data represents an extremely important asset for any organization, and enterprise users will face serious consequences if its confidential data is disclosed to their business competitors or the public. Thus, cloud users in the first place want to make sure that their data are kept confidential to outsiders, including the cloud provider and their potential competitors. This is the first data security requirement.

Data confidentiality is not the only security requirement. Flexible and fine-grained access control is also strongly desired in the service-oriented cloud computing model. A health-care information system on a cloud is required to restrict access of protected medical records to eligible doctors and a customer relation management system running on a cloud may allow

access of customer information to high-level executives of the company only. In these cases, access control of sensitive data is either required by legislation (e.g., HIPAA) or company regulations.

Access control is a classic security topic which dates back to the 1960s or early 1970s [9], and various access control models have been proposed since then. Among them, Bell-La Padula (BLP) [10] and BiBa [11] are two famous security models. To achieve flexible and fine-grained access control, a number of schemes [12]–[15] have been proposed more recently. Unfortunately, these schemes are only applicable to systems in which data owners and the service providers are within the same trusted domain. Since data owners and service providers are usually not in the same trusted domain in cloud computing, a new access control scheme employing attributed-based encryption [16] is proposed by Yu *et al.* [17], which adopts the so-called key-policy attribute-based encryption (KP-ABE) to enforce fine-grained access control. However, this scheme falls short of flexibility in attribute management and lacks scalability in dealing with multiple-levels of attribute authorities. We note that in contrast to KP-ABE, ciphertext-policy ABE (CP-ABE) [18] turns out to be well suited for access control due to its expressiveness in describing access control policies.

In this paper, we propose a hierarchical attribute-set-based encryption (HASBE) scheme for access control in cloud computing. HASBE extends the ciphertext-policy attribute-set-based encryption (CP-ASBE, or ASBE for short) scheme by Bobba *et al.* [19] with a hierarchical structure of system users, so as to achieve scalable, flexiblem and fine-grained access control.

The contribution of the paper is multifold. First, we show how HASBE extends the ASBE algorithm with a hierarchical structure to improve scalability and flexibility while at the same time inherits the feature of fine-grained access control of ASBE. Second, we demonstrate how to implement a full-fledged access control scheme for cloud computing based on HASBE. The scheme provides full support for hierarchical user grant, file creation, file deletion, and user revocation in cloud computing. Third, we formally prove the security of the proposed scheme based on the security of the CP-ABE scheme by Bethencourt *et al.* [18] and analyze its performance in terms of computational overhead. Lastly, we implement HASBE and conduct comprehensive experiments for performance evaluation, and our experiments demonstrate that HASBE has satisfactory performance.

The rest of the paper is organized as follows. Section II provides an overview on related work. Then we present our system model and assumptions in Section III. In Section IV, we describe in detail the construction of HASBE and show how it is used in access control of outsourced data in cloud computing. In Section V, we prove the security of HASBE and analyze its security by comparing with Yu *et al.*'s scheme. Then in Section VI, we analyze computation complexity of HASBE and evaluate its performance based on real implementation. Lastly, we conclude the paper in Section VII.

II. Related Work

In this section, we review the notion of attribute-based encryption (ABE), and provide a brief overview of the ASBE scheme by Bobba *et al.* After that, we examine existing access control schemes based on ABE.

A. Attribute-Based Encryption

The notion of ABE was first introduced by Sahai and Waters [20] as a new method for fuzzy identity-based encryption. The primary drawback of the scheme in [20] is that its threshold semantics lacks expressibility. Several efforts followed in the literature to try to solve the expressibility problem. In the ABE scheme, ciphertexts are not encrypted to one particular user as in traditional public key cryptography. Rather, both ciphertexts and users' decryption keys are associated with a set of attributes or a policy over attributes. A user is able to decrypt a ciphertext only if there is a match between his decryption key and the ciphertext. ABE schemes are classified into key-policy attribute-based encryption (KP-ABE) and ciphertext-policy attribute-based encryption (CP-ABE), depending how attributes and policy are associated with ciphertexts and users' decryption keys.

In a KP-ABE scheme [16], a ciphertext is associated with a set of attributes and a user's decryption key is associated with a monotonic tree access structure. Only if the attributes associated with the ciphertext satisfy the tree access structure, can the user decrypt the ciphertext. In a CP-ABE scheme [18], the roles of ciphertexts and decryption keys are switched; the ciphertext is encrypted with a tree access policy chosen by an encryptor, while the corresponding decryption key is created with respect to a set of attributes. As long as the set of attributes associated with a decryption key satisfies the tree access policy associated with a given ciphertext, the key can be used to decrypt the ciphertext. Since users' decryption keys are associated with a set of attributes, CP-ABE is conceptually closer to traditional access control models such as Role-Based Access Control (RBAC) [18]. Thus, it is more natural to apply CP-ABE, instead of KP-ABE, to enforce access control of encrypted data.

However, basic CP-ABE schemes (e.g., [18]) are far from enough to support access control in modern enterprise environments, which require considerable flexibility and efficiency in specifying policies and managing user attributes [19]. In a CP-ABE scheme, decryption keys only support user attributes that are organized logically as a single set, so users can only use all possible combinations of attributes in a single set issued in their keys to satisfy policies. To solve this problem, Bobba *et al.* [19] introduced ciphertext-policy attribute-set-based encryption (CP-ASBE or ASBE for short). ASBE is an extended form of CP-ABE which organizes user attributes into a recursive set structure. The following is an example of a key structure of depth 2, which is the depth of the recursive set structure:

$$\{\mathrm{Dept}:\mathrm{CS},\mathrm{Role}:\mathrm{Grad}-\mathrm{Student},\\ \{\mathrm{CourseID}:101,\mathrm{Role}:\mathrm{TA}\},\\ \{\mathrm{CourseID}:525,\mathrm{Role}:\mathrm{Grad}-\mathrm{Student}\}\}.$$

The above example represents a key structure assigned to a graduate student in CS department of a university, who is the TA for course 101 and has enrolled in course 525. It can be seen that the same attribute can be assigned multiple values, e.g., the attribute "Role" is assigned value "TA" and "Grad-Student" in different sets. This feature renders ASBE more versatile and flexible in supporting many practical scenarios. In this example, the graduate student holding such a private key should not be able to combine the attribute "Role: TA" with "CourseID:

525" so as to access course grades of other students who enroll in course 525. Such a feature cannot be implemented with the original CP-ABE algorithm.

ASBE can enforce dynamic constraints on combining attributes to satisfy a policy, which provides great flexibility in access control. In the recursive attribute set assigned to a user, attributes from the same set can be combined freely, while attributes from different sets can only be combined with the help of translating items, whose function will be explained later. Consider attributes for students derived from courses they have taken. Every student has a set of attributes $(\mathrm{Course}, \mathrm{Year}, \mathrm{Grade})$ for each course he has taken. We want to have a policy "*Students who took a course that satisfies* $300 \leq \mathrm{Course} < 400$ *and* $\mathrm{Year} \geq 2009$ *and* $\mathrm{Grade} > 3$." Enforcing such a policy with CP-ABE is difficult, since a student could have taken multiple courses and obtained different grades in them. The encryptor will have to ensure the student cannot select and combine attributes from different sets to circumvent the policy. In [19], several possible solutions with plain CP-ABE are described, but none of them is satisfactory. However, using ASBE, we can solve the problem simply by assigning multiple values to the group of attributes in different sets. For each course the student has taken, he gets a separate set of values for the attributes $(\mathrm{Course}, \mathrm{Grade}, \mathrm{Year})$. In this way, ASBE can enforce efficient ciphertext policy encryption for situations where existing ABE schemes are inefficient.

Furthermore, ASBE's capability of assigning multiple values to the same attribute enables it to solve the user revocation problem efficiently, which is difficult in CP-ABE. The revocation problem can be solved easily by assigning different expiration times.

The above desirable feature and the recursive key structure is implemented by four algorithms, **Setup**, **KeyGen**, **Encrypt**, and **Decrypt**:

$\mathrm{Setup}(d)$. Here d is the depth of key structure. Take as input a depth parameter d. It outputs a public key $\mathbf{PK}$ and master secret key $\mathbf{MK}$.

$\mathrm{KeyGen}(\mathbf{MK}, u, \mathbb{A})$ Take as input the master secret key $\mathbf{MK}$, the identity of user u, and a key structure $\mathbb{A}$. It outputs a secret key $\mathbf{SK}_u$ for user u.

$\mathrm{Encrypt}(\mathbf{PK}, M, \mathcal{T})$ Take as input the public key $\mathbf{PK}$, a message M, and an access tree $\mathcal{T}$. It outputs a ciphertext $\mathbf{CT}$.

$\mathrm{Decrypt}(\mathbf{CT}, \mathbf{SK}_u)$. Take as input a ciphertext $\mathbf{CT}$ and a secret key $\mathbf{SK}_u$ for user u. It outputs a message m. If the key structure $\mathbb{A}$ associated with the secret key $\mathbf{SK}_u$ satisfies the access tree $\mathcal{T}$, associated with the ciphertext $\mathbf{CT}$, then m is the original correct message M. Otherwise, m is null.

These algorithms are essentially similar to those of CP-ABE, except some extensions to support recursive key structure. The public key and the master key of ASBE are extended from CP-ABE to have components supporting recursive key structure. For depth d, the corresponding public key component is h_d and f_d. The master key is extended by adding a new secret exponent β_d for depth d. The generated private keys are also different in ASBE and CP-ABE. There are translating components that enable attributes translation between different key sets.

The missing part of ASBE is the delegation algorithm, which is used in our proposed scheme to construct the hierarchical structure. We adopt the same four algorithms of ASBE, and extend ASBE by proposing a new delegation algorithm.

B. Access Control Solutions for Cloud Computing

The traditional method to protect sensitive data outsourced to third parties is to store encrypted data on servers, while the decryption keys are disclosed to authorized users only. However, there are several drawbacks about this trivial solution. First of all, such a solution requires an efficient key management mechanism to distribute decryption keys to authorized users, which has been proven to be very difficult. Next, this approach lacks scalability and flexibility; as the number of authorized users becomes large, the solution will not be efficient any more. In case a previously legitimate user needs to be revoked, related data has to be re-encrypted and new keys must be distributed to existing legitimate users again. Last but not least, data owners need to be online all the time so as to encrypt or re-encrypt data and distribute keys to authorize users.

ABE turns out to be a good technique for realizing scalable, flexible, and fine-grained access control solutions. Yu *et al.* [17] proposed an access control mechanism based on KP-ABE for cloud computing, together with a re-encryption technique for efficient user revocation. This scheme enables a data owner to delegate most of the computational overhead to cloud servers. The use of KP-ABE provides fine-grained access control gracefully. Each file is encrypted with a symmetric data encryption key (DEK), which is in turn encrypted by a public key corresponding to a set of attributes in KP-ABE, which is generated according to an access structure. The encrypted data file is stored with the corresponding attributes and the encrypted DEK. If the associated attributes of a file stored in the cloud satisfy the access structure of a user's key, then the user is able to decrypt the encrypted DEK, which is used in turn to decrypt the file.

The first problem with Yu *et al.*'s scheme is that the encryptor is not able to decide who can decrypt the encrypted data except choosing descriptive attributes for the data, and has no choice but to trust the key issuer. Furthermore, KP-ABE is not naturally suitable to certain applications. An example of such applications is a type of sophisticated broadcast encryption, where users are described by various attributes and the one whose attributes match a policy associated with a ciphertext can decrypt the ciphertext. For such an application, a better choice is CP-ABE.

Wang *et al.* [21] proposed hierarchical attribute-based encryption (HABE) to achieve fine-grained access control in cloud storage services by combining hierarchical identity-based encryption (HIBE) and CP-ABE. This scheme also supports fine-grained access control and fully delegating computation to the cloud providers. However, HABE uses disjunctive normal form policy and assumes all attributes in one conjunctive clause are administrated by the same domain master. Thus the same attribute may be administrated by multiple domain masters according to specific policies, which is difficult to implement in practice. Furthermore, compared with ASBE, this scheme cannot support compound attributes efficiently and does not support multiple value assignments.

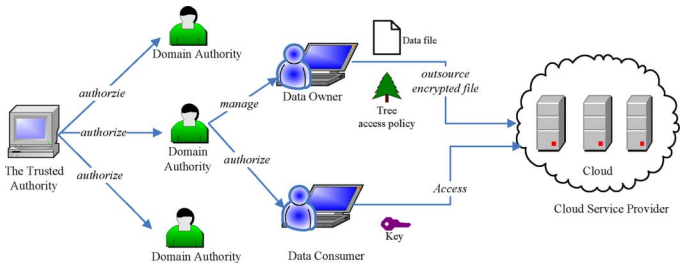

Fig. 1. System model.

III. System Model and Assumptions

A. System Model

As depicted in Fig. 1, the cloud computing system under consideration consists of five types of parties: a *cloud service provider*, *data owners*, *data consumers*, a number of *domain authorities*, and a *trusted authority*.

The cloud service provider manages a cloud to provide data storage service. Data owners encrypt their data files and store them in the cloud for sharing with data consumers. To access the shared data files, data consumers download encrypted data files of their interest from the cloud and then decrypt them. Each data owner/consumer is administrated by a domain authority. A domain authority is managed by its parent domain authority or the trusted authority. Data owners, data consumers, domain authorities, and the trusted authority are organized in a hierarchical manner as shown in Fig. 1.

The trusted authority is the root authority and responsible for managing top-level domain authorities. Each top-level domain authority corresponds to a top-level organization, such as a federated enterprise, while each lower-level domain authority corresponds to a lower-level organization, such as an affiliated company in a federated enterprise. Data owners/consumers may correspond to employees in an organization. Each domain authority is responsible for managing the domain authorities at the next level or the data owners/consumers in its domain.

In our system, neither data owners nor data consumers will be always online. They come online only when necessary, while the cloud service provider, the trusted authority, and domain authorities are always online. The cloud is assumed to have abundant storage capacity and computation power. In addition, we assume that data consumers can access data files for reading only.

B. Security Model

We assume that the cloud server provider is untrusted in the sense that it may collude with malicious users (short for data owners/data consumers) to harvest file contents stored in the cloud for its own benefit.

In the hierarchical structure of the system users given in Fig. 1, each party is associated with a public key and a private key, with the latter being kept secretly by the party. The trusted authority acts as the root of trust and authorizes the top-level domain authorities. A domain authority is trusted by its subordinate domain authorities or users that it administrates, but may try to get the private keys of users outside its domain. Users may try to access data files either within or outside the scope of their access privileges, so malicious users may collude with each other to get sensitive files beyond their privileges. In addition, we assume that communication channels between all parties are secured using standard security protocols, such as SSL.

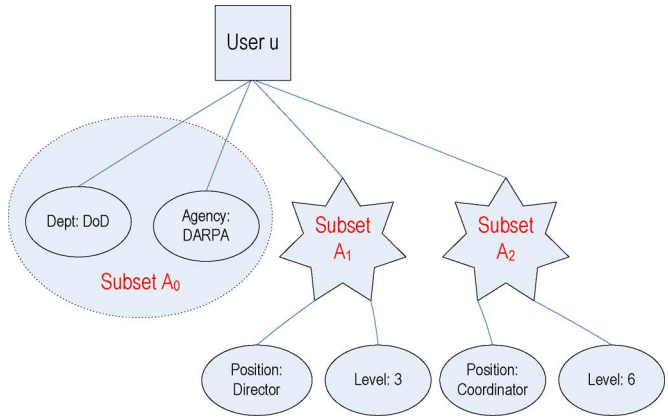

Fig. 2. Example key structure.

IV. Our Construction

In this section, we first present our HASBE scheme, which extends the ASBE algorithm with a hierarchical user structure. We then show how HASBE is applied for hierarchical user grant, data file creation, file access, user revocation, and file deletion.

A. Preliminaries

Bilinear Maps: Let $\mathbb{G}$, $\mathbb{G}_1$ be cyclic (multiplicative) groups of prime order p. Let g be a generator of $\mathbb{G}$. Then e: $\mathbb{G}\times\mathbb{G} \rightarrow \mathbb{G}_1$ is a bilinear map if it has the following properties:.

- *Bilinearity*: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
- *Nondegeneracy*: $e(g, g) \neq 1$.

$\mathbb{G}$ is called a bilinear group if the group operation and the bilinear map e are both efficiently computable.

In our HASBE scheme, a data encryptor specifies an access structure for a ciphertext which is referred to as the ciphertext policy. Only users with decryption keys whose associated attributes, specified in their key structures, satisfy the access structure can decrypt the ciphertext.

Key Structure: We use a recursive set based key structure as in [19] where each element of the set is either a set or an element corresponding to an attribute. The *depth* of the key structure is the level of recursions in the recursive set, similar to definition of depth for a tree. For a key structure with depth 2, members of the set at depth 1 can either be attribute elements or sets but members of a set at depth 2 may only be attribute elements. Consider the example shown in Fig. 2, where $\{\text{Dept}: \text{DoD}, \text{Agency}: \text{DARPA}, \text{Position}: \text{Director}, \text{Level}: 3\}$, $\{\text{Position}: \text{Coordinator}, \text{Level}: 6\}\}$ is a key structure of depth 2. It represents the attributes of a person who is both a director of level 3 for a unit and a coordinator of level 6 for another unit in the Defense Advanced Research Projects Agency (DARPA) of the Department of Defense (DoD).

The key structure defines unique labels for sets in it. For key structures of depth 2, just an index of the sets at depth 2 is sufficient to uniquely identify the sets. Thus if there are m sets

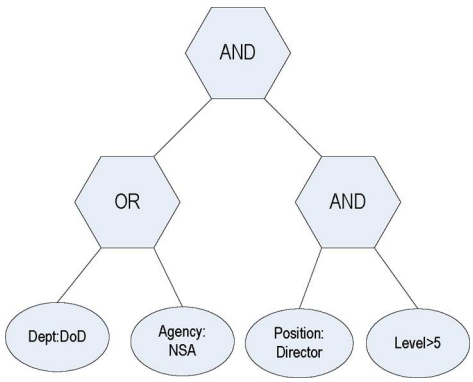

Fig. 3. Example access structure.

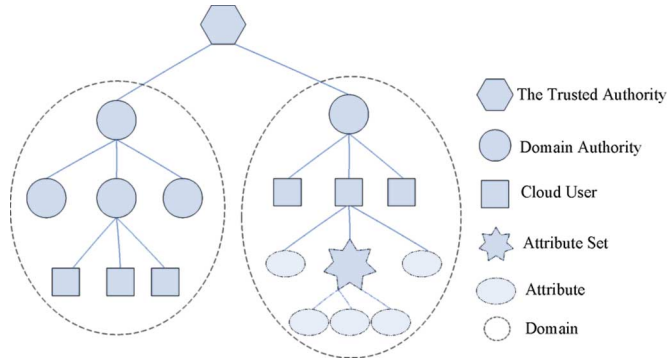

Fig. 4. Hierarchical structure of system users.

at depth 2 then a unique index i where $1 \leq i \leq m$ is assigned to each set. The set at depth 1 is referred to as set 0. Using this convention, a key structure of depth 2 can be represented as $\mathbb{A} = \{A_0, A_1, \ldots, A_m\}$, where A_0 is the set at depth 1 while A_i is the ith set at depth 2, for $1 \leq i \leq m$. In the key structure in Fig. 2, $\{\text{Dept : DoD, Agency : DARPA}\}$ corresponds to A_0, $\{\text{Position : Director, Level : 3}\}$ and $\{\text{Position : Coordinator, Level : 6}\}$ correspond to A_1 and A_2, respectively. Individual attributes inherit the label of the set they are contained in and are uniquely defined by the combination of their name and their inherited label. For example, attribute Dept : DoD is defined as $(0, \text{Dept : DoD})$. When trying to satisfy a given policy, a user may only use attribute elements within a set, but may not combine attributes across the sets by default. However, if the encryptor has designated *translating nodes* in an access structure, users can combine attributes from multiple sets to satisfy the access structure, as will be explained later in the scheme construction as well as in [19].

Access Structure: In our scheme, we use the same tree access structure as in [19]. In the tree access structure, leaf nodes are attributes and nonleaf nodes are threshold gates. Each nonleaf node is defined by its children and a threshold value. Let num_x denote the number of children and k_x the threshold value of node x. An example of the access tree structure is shown in Fig. 3, where the threshold values for "AND" and "OR" are 2 and 1, respectively.

The above access structure demands that only a director in DoD or NSA of level larger than 5 can access the data files protected by the access policy. In CP-ABE schemes, a person who has private keys corresponding to attributes on the key structure shown in Fig. 2 would be able to access the data files, which compromises the security of the access policy in Fig. 3. Such problems are effectively prevented using attribute-set-based encryption which forbids combining attributes across multiple sets.

Let $\mathcal{T}_x$ be the access structure rooted at node x and $\mathcal{T}$ be the access structure rooted at the root node R. Without loss of generality, we consider key structure of depth 2, $\mathbb{A} = \{A_0, A_1, \ldots, A_m\}$, where $A_i (0 \leq i \leq m)$ is the ith attribute set and i is the label. We say that $\mathbb{A}$ satisfies $\mathcal{T}$ if only if a function $\mathcal{T}(\mathbb{A})$ returns a nonempty set S of labels. The function $\mathcal{T}(\mathbb{A})$ is computed recursively and will be introduced in the encryption algorithm later. $\mathbb{A}$ is said to satisfy $\mathcal{T}$ if it contains at least one set $A_i (0 \leq i \leq m)$ that has all the attributes needed to satisfy $\mathcal{T}$ and that the attributes belonging to multiple sets in $\mathbb{A}$ cannot be combined to satisfy $\mathcal{T}$, except when there are designated translating nodes in $\mathcal{T}$. If node x is a translating node in $\mathcal{T}$, then if the attribute elements used to satisfy the predicate represented by the subtree rooted at x belong to a different set in $\mathbb{A}$ than those used to satisfy the predicates represented by the siblings of x, the decrypting user is able to combine them to satisfy the predicate represented by the parent node of x.

Several functions are defined for the purpose of dealing with the access structure. We define $\text{parent}(x)$ as the parent node of x and $\text{index}(x)$ as the index number of node x. The function $\mathbf{att}(x)$ is defined only if x is a leaf node and denotes the attribute associated with the leaf node x in the tree.

B. HASBE Scheme

The proposed HASBE scheme seamlessly extends the ASBE scheme to handle the hierarchical structure of system users in Fig. 4.

Recall that our system model consists of a trusted authority, multiple domain authorities, and numerous users corresponding to data owners and data consumers. The trusted authority is responsible for generating and distributing system parameters and root master keys as well as authorizing the top-level domain authorities. A domain authority is responsible for delegating keys to subordinate domain authorities at the next level or users in its domain. Each user in the system is assigned a key structure which specifies the attributes associated with the user's decryption key.

We are now ready to describe the main operations of HASBE: *System Setup*, *Top-Level Domain Authority Grant*, *New Domain Authority/User Grant*, *New File Creation*, *User Revocation*, *File Access*, and *File Deletion*.

System Setup: The trusted authority calls the Setup algorithm to create system public parameters $\mathbf{PK}$ and master key $\mathbf{MK}_0$. $\mathbf{PK}$ will be made public to other parties and $\mathbf{MK}_0$ will be kept secret.

$\text{Setup}(d = 2) \rightarrow (\mathbf{PK}, \mathbf{MK}_0)$. Here d is the depth of the key structure. We describe the HASBE scheme for key structures of depth 2, and it can be extended to any depth d. The algorithm selects a bilinear group $\mathbb{G}$ of prime order p with generator g and then chooses random exponents $\alpha, \beta_i \in Z_p, \forall i\{1, 2\}$. To

support key structure of depth d, i will range from 1 to d. This algorithm sets the public key and master key as follows:

$$\mathbf{PK} = \Big(\mathbb{G}, g, h_1 = g^{\beta_1}, \quad f_1 = g^{\frac{1}{\beta_1}}, \\ h_2 = g^{\beta_2}, \quad f_2 = g^{\frac{1}{\beta_2}}, e(g,g)^{\alpha}\Big)$$
$$\mathbf{MK}_0 = (\beta_1, \beta_2, g^{\alpha}).$$

Top-Level Domain Authority Grant: A domain authority is associated with a unique ID and a recursive attribute set $\mathbb{A} = \{A_0, A_1, \ldots, A_m\}$, where $A_i = \{a_{i,1}, a_{i,2}, \ldots, a_{i,n}\}$ with $a_{i,j}$ being the jth attribute in A_i and n_i being the number of attributes in A_i. When a new top-level domain authority, i.e., DA_i, wants to join the system, the trusted authority will first verify whether it is a valid domain authority. If so, the trusted authority calls $\mathrm{CreateDA}$ to generate the master key for DA_i. After getting the master key, DA_i can authorize the next level domain authorities or users in its domain.

$\mathrm{CreateDA}(\mathbf{PK}, \mathbf{MK}_0, \mathbb{A})$. This algorithm creates the master key for top-level DA_i. It selects a unique number $r^{\{u\}}$ for the domain authority, which is also for the set A_0, and selects m random numbers $r_i^{\{u\}} \in \mathbb{Z}_p$, one for each set $\mathrm{A}_i \in \mathbb{A}$. Furthermore, it picks a random number $r_{i,j}^{\{u\}}$ for each $a_{i,j}, 0 \le i \le m, 1 \le j \le n_i$. It computes the master key for DA_i as follows:

$$\mathbf{MK}_i = \Bigg(\mathbb{A}, D = g^{\frac{\left(\alpha + r^{\{u\}}\right)}{\beta_1}}, D_{i,j} = g^{r_i^{\{u\}}} \cdot H(a_{i,j})^{r_{i,j}^{\{u\}}}, \\ D'_{i,j} = g^{r_{i,j}^{\{u\}}} \text{ for } 0 \le i \le m,\ 1 \le j \le n_i, \\ E_i = g^{\frac{\left(r^{\{u\}} + r_i^{\{u\}}\right)}{\beta_2}} \text{ for } 1 \le i \le m\Bigg).$$

In the above master key, E_i is for translation from $r_i^{\{u\}}$ of A_i to $r^{\{u\}}$ of A_0 at the *translating node*. Elements E_i and $E_{i'}$ can be used as $E_i/E_{i'}$ to translate $r_{i'}^{\{u\}}$ to $r_i^{\{u\}}$ at the *translating nodes*, we will give the details later in the $\mathrm{Decrypt}$ algorithm.

New Domain Authority/User Grant: When a new user, denoted as u, or a new subordinate domain authority, denoted as DA_{i+1}, wants to join the system, the administrating domain authority, denoted as DA_i, will first verify whether the new entity is valid. If true, DA_i assigns the new entity a key structure $\widetilde{\mathbb{A}}$ corresponding to its role and a unique ID. Note that $\widetilde{\mathbb{A}}$ is a subset of $\mathbb{A}$, where $\mathbb{A}$ is the key structure of DA_i. In $\widetilde{\mathbb{A}}$, every element is labeled the same as it is in $\mathbb{A}$. For example, $\mathbb{A} = \{\mathrm{University:\ A},\ \mathrm{College:\ B},\ \{\mathrm{Course:\ 100}, \mathrm{Grade:\ 80}\}, \{\mathrm{Course:\ 101},\ \mathrm{Grade:\ 85}\}\}$ and $\widetilde{\mathbb{A}} = \{\mathrm{University:\ A},\ \{\mathrm{Course:\ 101},\ \mathrm{Grade:\ 85}\}\}$, then $\{\mathrm{Course:\ 101}, \mathrm{Grade:\ 85}\}$ is labeled as set A_2 in both $\mathbb{A}$ and $\widetilde{\mathbb{A}}$, and $\mathrm{Course:\ 101}$ is labeled as $(2, \mathrm{Course:\ 101})$.

For a new user u, DA_i calls $\mathrm{CreateUser}(\mathbf{MK}_i, u, \widetilde{\mathbb{A}})$ to generate the secret key for this user. Otherwise, if it is a new domain authority DA_{i+1}, DA_i calls $\mathrm{CreateUser}(\mathbf{MK}_i, \mathrm{DA}_{i+1}, \widetilde{\mathbb{A}})$ to generate the master key for DA_{i+1}. Then DA_{i+1} can authorize the lower level domain authorities or users in its domain.

$\mathrm{CreateUser}(\mathbf{MK}_i, u, \widetilde{\mathbb{A}})$. This algorithm uses the master key of DA_i, which is for the key structure $\mathbb{A}$, and a new key structure $\widetilde{\mathbb{A}}$, which is a set of $\mathbb{A}$. The master key of DA_i is in the form $\mathbf{MK}_i = (\mathbb{A}, D, D_{i,j}, D'_{i,j}$ for $0 \le i \le m, 1 \le j \le n_i, E_i$ for $1 \le i \le m)$. As in the $\mathrm{CreateDA}$ algorithm, this algorithm randomly chooses a unique number $\widetilde{r}^{\{u\}}$ for each user or domain authority, a random number $\widetilde{r}_i^{\{u\}}$ for each set $\mathrm{A}_i \in \widetilde{\mathbb{A}}$, and a random number $\widetilde{r}_{i,j}^{\{u\}}$ for each $a_{i,j} \in \widetilde{\mathbb{A}}$. Then it computes the new secret key as

$$\mathbf{SK}_u \text{ or } \mathbf{MK}_{i+1} \\ = \Big(\widetilde{\mathbb{A}}, \widetilde{D} = D \cdot f_1^{\widetilde{r}^{\{u\}}}, \widetilde{D}_{i,j} = D_{i,j} \cdot g^{\widetilde{r}_i^{\{u\}}} \cdot H(a_{i,j})^{\widetilde{r}_{i,j}^{\{u\}}}, \\ \widetilde{D}'_{i,j} = D'_{i,j} \cdot g^{\widetilde{r}_{i,j}^{\{u\}}} \text{ for } a_{i,j} \in \widetilde{\mathbb{A}}, \\ \widetilde{E}_i = E_i \cdot f_2^{\widetilde{r}^{\{u\}} + \widetilde{r}_i^{\{u\}}} \text{ for } A_i \in \widetilde{\mathbb{A}}\Big).$$

The new secret key $\mathbf{SK}_u$ or $\mathbf{MK}_{i+1}$ is a secret key for the key structure $\widetilde{\mathbb{A}}$. Because the algorithm rerandomizes the key, a delegated key is equivalent to one received directly from the trusted authority.

New File Creation: To protect data stored on the cloud, a data owner first encrypts data files and then stores the encrypted data files on the cloud. As in [16], each file is encrypted with a symmetric data encryption key DEK, which is in turn encrypted with HASBE. Before uploading to the cloud, a data file is processed by the data owner as follows:

- Pick a unique ID for this data file.
- Randomly choose a symmetric data encryption key $\mathrm{DEK} \xleftarrow{R} \kappa$, where κ is the key space, and encrypt the data file using DEK.
- Define a tree access structure $\mathcal{T}$ for the file and encrypt DEK with $\mathcal{T}$ using algorithm $\mathrm{Encrypt}(\mathbf{PK}, \mathrm{DEK}, \mathcal{T})$ of HASBE which returns ciphetext CT.

Finally, the encrypted data file is stored on the cloud in the format as shown in Fig. 5.

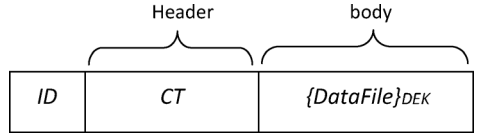

Fig. 5. Format of a data file on the cloud.

$\mathrm{Encrypt}(\mathbf{PK}, M, \mathcal{T})$. M is the message to encrypt. In the *New File Creation* operation, M is the DEK of a data file. $\mathcal{T}$ is the tree access structure. Encrypt algorithm is the same as that of ASBE [19]. The algorithm associates a polynomial q_x with each node x in the tree $\mathcal{T}$, which is chosen randomly in a top-down manner from the root node R. For every node x in $\mathcal{T}$, the degree of q_x is set to be one less than the threshold value of x and denoted as d_x. If x is a leaf node, then d_x is set to 0. For each nonroot node x, $q_x(0) = q_{\mathrm{parent}(x)}(\mathrm{index}(x))$. The other d_x points of q_x are randomly chosen. For the root node R, $q_R(0) = s$, where $s \in Z_p$ is a random number, and the other d_R points of q_R are randomly selected. This algorithm computes the Ciphetext as follows:

$$\mathbf{CT} = +\Big(\mathcal{T}, \widetilde{C} = M \cdot e(g,g)^{\alpha \cdot s}, C = h_1^s, \bar{C} = h_2^s, \forall y \in Y : \\ C_y = g^{q_y(0)}, C'_y = H\left(\mathbf{att}(y)\right)^{q_y(0)}, \\ \forall x \in X : \hat{C}_x = h_2^{q_x(0)}\Big)$$

where Y denotes the set of leaf nodes in $\mathcal{T}$, X denotes the set of translating nodes in the access tree $\mathcal{T}$.

User Revocation: Whenever there is a user to be revoked, the system must make sure the revoked user cannot access the associated data files any more. One way to solve this problem is to re-encrypt all the associated data files used to be accessed by the revoked user, but we must also ensure that the other users who still have access privileges to these data files can access them correctly.

HASBE inherits the advantage of ASBE in efficient user revocation. We add an attribute $\mathrm{expiration\_time}$ to a user's key, which indicates the time until which the key is considered to be valid. Then the policy associated with data files can include a check on the $\mathrm{expiration\_time}$ attribute as a numerical comparison. For example, assuming a user u has a key with $\mathrm{expiration\_time}X$ and a data file whose access policy is associated with $\mathrm{expiration\_time}Y$, then u can decrypt this data file only when $X \geq Y$ and the rest of the policy matches u's attributes. This numeric comparison of attributes can be implemented by the "bag of bits" as in [18]. In practice, the validity period of sensitive attributes must be kept small to reduce the *window of vulnerability* when a key is compromised, for example, a day, a week, or a month [19]. With this feature, we allow multiple value assignments to the $\mathrm{expiration\_time}$ attribute so as to add a new expiration value to the existing key. In this way, we can update user's key without entire key regenerating and redistributing at the end of expiration time. On the other hand, the data owner can change the policy over data files by updating the $\mathrm{expiration\_time}$ attribute associated with the leaf node in the access tree. The update of user's key and re-encryption of data files can be done as follows:

Key Update. Suppose that there is a user u, who is administrated by the domain authority DA_i. DA_i maintains some state information about u's key and adds a new value of $\mathrm{expiration\_time}$ to u's existing key when it wants to update u's key. Then DA_i computes the secret key components corresponding to the $\mathrm{expiration\_time}$ attribute and sends them to u. Transmission of the secret key components to the user can be accomplished with an out-of-band channel between DA_i and the user u. While DA_i is required to maintain some state information about user's key, DA_i avoids the need to generate and distribute the entire keys on a frequent basis. This reduces the workload on DA_i and saves considerable computing resources.

Data Re-encryption. When the data owner wants to re-encrypt a data file, he changes the value of the $\mathrm{expiration\_time}$ attribute in the key policy and computes the new ciphertext components C_y and C'_y, where y is the leaf node on the access tree corresponding the $\mathrm{expiration\_time}$ attribute. Then the data owner sends these new ciphertext components to the cloud and the cloud service provider can re-encrypt the data file by simply updating these ciphertext components. So when re-encrypting a data file, the data owner just needs to compute the ciphertext components associated with the $\mathrm{expiration\_time}$ attribute while other parts of the ciphertext remain unchanged, which effectively reduces the workload of the data owner. Furthermore, in this process the cloud just knows the two ciphertext components and can not get the plaintext of the data file.

File Access: When a user sends request for data files stored on the cloud, the cloud sends the corresponding ciphertexts to the user. The user decrypts them by first calling $\mathrm{Decrypt}(\mathrm{CT}, \mathrm{SK}_u)$ to obtain DEK and then decrypt data files using DEK. $\mathrm{Decrypt}(\mathrm{CT},\ \mathrm{SK}_u)$ algorithm is as follows:

$\mathrm{Decrypt}(\mathrm{CT}, \mathrm{SK}_u)$. This algorithm accepts ciphetext CT and user u's key structure as input. The algorithm first calls $\mathcal{T}(\mathbb{A})$ to verify whether the key structure $\mathbb{A}$ in SK_u satisfies the tree access structure $\mathcal{T}$ associated with the CT. The function $\mathcal{T}(\mathbb{A})$ is performed recursively. For each node x in $\mathcal{T}$, there is a set S_x of labels returned by $\mathcal{T}_x(\mathbb{A})$. If $\mathbb{A}$ does not satisfy $\mathcal{T}$, the algorithm returns null; otherwise the algorithm picks one i from the set returned by $\mathcal{T}(\mathbb{A})$, and calls function $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}_u, t, i)$ on the root node of $\mathcal{T}$, where t is a node from $\mathcal{T}$. $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}_u, t, i)$ is defined as follows:

If t is a leaf node, and if $\mathbf{att}(t) \notin A_i$, where $A_i \in \mathbb{A}$, then $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}_u, t, i) = \mathrm{null}$. If $\mathbf{att}(t) = a_{i,j} \in A_i$, where $A_i \in \mathbb{A}$, then $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}_u, t, i) = e(D_{i,j}, C_t)/e(D'_{i,j}, C'_t) = e(g,g)^{r_i^{\{u\}}} \cdot q_t(0)$.

If t is a nonleaf node, then $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}_u, t, i)$ is defined as follows:

- Let B_t be an arbitrary k_t sized set of child nodes z such that $z \in B_t$ only if (1) label $i \in S_z$ or (2) label $i' \in S_z$ for some $i' \neq i$ and z is a *translating node*. If no such set exists then return null.
- For each node $z \in B_t$, if $i \in S_z$, then call $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}_u, z, i)$ and store output in F_z.
- For each node $z \in B_t$, if $i' \in S_z$ and $i' \neq i$, then call $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}_u, z, i')$ and store output in F'_z. Then if $i \neq 0$, translate F'_z to F_z as follows:

$$F_z = e(\hat{C}_z, E_i/E_{i'}) \cdot F'_z = e(g,g)^{r_i^{\{u\}} \cdot q_z(0)}.$$

Otherwise, if $i = 0$, then translate F'_z to F_z as follows:

$$F_z = \frac{e(\hat{C}_z, E_{i'})}{F'_z} = e(g,g)^{r^{\{u\}}} \cdot q_z(0).$$

- Compute F_t using polynomial interpolation as follows: $F_t = \prod_{z \in B_t} F_z^{\triangle_{k,B'_z}(0)}$, where $k = index(z), B'_z = \{index(z) : z \in B_t\}$. So when $i = 0$, $F_t = e(g,g)^{r^{\{u\}} \cdot q_t(0)}$, else when $i \neq 0$, $F_t = e(g,g)^{r_i^{\{u\}} \cdot q_t(0)}$.

So the function $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}_u, R, i)$ on the root node R returns F_R. If $i = 0$, then $F = F_R = e(g,g)^{r^{\{u\}} \cdot q_R(0)} = e(g,g)^{r^{\{u\}} \cdot s}$. If $i \neq 0$, then $F_R = e(g,g)^{r_i^{\{u\}} \cdot s}$ and $F = e(\hat{C}_r, E_i)/F_R = e(g,g)^{r^{\{u\}} \cdot s}$.

Then the message M can be computed as $M = \tilde{C} \cdot F/e(C, D)$.

File Deletion: Encrypted data files can be deleted only at the request of the data owner. To delete an encrypted data file, the data owner sends the file's unique ID and its signature on this ID to the cloud. Only upon successful verification of the data owner and the request, the cloud deletes the data file.

V. Security Proof and Discussion

A. Security Proof

Though HASBE is extended from ASBE by Bobba *et al.* with a hierarchical structure using a delegation algorithm similar to the one described in the CP-ABE scheme by Bethencourt *et al.*, we do not use the proof technique by Bobba *et al.* Instead, we prove the security of our scheme directly based on the security of CP-ABE. We show that if there are any vulnerabilities in the proposed scheme, these vulnerabilities can be used to break CP-ABE. Thus, HASBE is expected to have the same security property as CP-ABE, which has been proven to be secure under the generic bilinear group model and the random oracle model.

A generic security model to be defined below describes interactions between an adversary and an encryption algorithm like HASBE or CP-ABE. Identical to the model used in CP-ABE, the security model allows the adversary to query for any private keys that cannot be used to decrypt the challenge ciphertext. In CP-ABE and HASBE the ciphertexts are associated with access structures and the private keys are identified with attributes. Thus, the security model requires that the adversary chooses to be challenged on an encryption to an access structure $\mathbb{A}^*$ and can ask for any private key S such that S does not satisfy $\mathbb{A}^*$.

1) Formal Security Model: Before giving a formal proof for the proposed scheme, we first describe the formal security model for ciphertext-policy ABE schemes. In this model, the adversary will choose to be challenged on an encryption to an access structure $\mathbb{A}^*$ and can ask for any private key S such that S does not satisfy $\mathbb{A}^*$. The formal security model is defined as follows between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

- **Setup**. The challenger runs the Setup algorithm and gives the public parameters, PK to the adversary.
- **Phase 1**. The adversary makes repeated private key queries corresponding to sets of attributes $S_1, \ldots, S_{q_1}$. The challenger responds by running algorithm $\mathrm{CreateDA}$ (Top-level domain) to generate the private key SK_i corresponding to the attribute set S_i. Or else, the adversary makes private key queries for a lower-level domain authority (MK_{i+1}) or end users (SK_u) with the private key MK_i of an upper level domain authority. The challenger responds by running algorithm $\mathrm{CreateUser}$ to generate the private key.
- **Challenge**. The adversary submits two equal length messages M_0 and M_1. In addition, the adversary gives a challenge access structure $\mathbb{A}^*$ such that none of the sets $S_1, \ldots, S_{q_1}$ from Phase 1 satisfy the access structure. The challenger flips a random coin b, and encrypts M_b under $\mathbb{A}^*$. The ciphertext $\mathbf{CT}^*$ is given to the adversary.
- **Phase 2**. Phase 1 is repeated with the restriction that none of the sets of attributes $S_{q_1+1}, \ldots, S_q$ satisfy the access structure corresponding to the challenge.
- **Guess**. The adversary outputs a guess b' of b.

The advantage of the adversary $\mathcal{A}$ in this game is defined as $\Pr[b' = b] - (1/2)$.

Definition 1: A ciphertext-policy ABE scheme is secure if all polynomial time adversaries have at most a negligible advantage in the above game.

Theorem 1: Suppose there is no polytime adversary who can break the security of CP-ABE with nonnegligible advantage; then there is no polytime adversary who can break our system with nonnegligible advantage.

Proof: Suppose we have an adversary $\mathcal{A}$ with nonnegligible advantage against our proposed scheme. Using $\mathcal{A}$, we show how to build an adversary, $\mathcal{B}$, that breaks the CP-ABE scheme with nonnegligible advantage. The adversary $\mathcal{B}$ can play a similar game with the CP-ABE scheme. The CP-ABE security model [18] is also composed of four steps: Setup, Phase 1, Challenge, Phase 2 and Guess. That is to say, $\mathcal{B}$ can make private queries during the game to obtain private keys in the CP-ABE scheme.

- **Initialization**. The adversary $\mathcal{B}$ takes the public key of CP-ABE $\mathrm{PK}' = \{\mathbb{G}, g, h = g^\beta, f = g^{1/\beta}, e(g,g)^\alpha\}$, and the corresponding private key (β, g^α) is unknown to the adversary.
- **Setup**. The adversary $\mathcal{B}$ selects a random number $t \in Z_p$, and computes the HASBE public parameters from PK' as $\mathrm{PK} = \{\mathbb{G}, g, h_1 = g^\beta, f_1 = g^{1/\beta}, h_2 = g^{t\beta}, f_2 = g^{1/(t\beta)}, e(g,g)^\alpha\}$. That is, the adversary $\mathcal{B}$ sets $\beta_1 = \beta$ and $\beta_2 = t \cdot \beta$. Then the public key PK is given to the adversary.
- **Phase 1**. In this phase, $\mathcal{B}$ answers private key queries. Suppose the adversary $\mathcal{B}$ is given a private key query for a set S where S does not satisfy $\mathbb{A}^*$. In order to answer the query, $\mathcal{B}$ makes a private key query to CP-ABE challenger for the same set S twice. As a result, $\mathcal{B}$ obtains two different private keys:

$$\mathrm{SK} = \Big(D = g^{(\alpha+r)/\beta}, \forall a_j \in S : D_j = g^r \cdot H(a_j)^{r_j}, D'_j = g^{r_j}\Big),$$

$$\mathrm{SK}' = \Big(D = g^{(\alpha+r')/\beta}, \forall a_j \in S : D_j = g^{r'} \cdot H(a_j)^{r'_j}, D'_j = g^{r'_j}\Big)$$

where a_j's are attributes from S, and r, r', r_j, r'_j are random numbers in Z_p.

From SK and SK′, $\mathcal{B}$ can obtain $g^{(r-r')/\beta}$ by dividing D in SK with D in SK′. $\mathcal{B}$ selects random number $t_i, t_{i,j} \in Z_p$, and let $r^* = t_i - r'$ and $r'' = t_{i,j} - r'_j$. Then $\mathcal{B}$ can derive the private key requested by $\mathcal{A}$ as $\mathbf{MK}^* = (D = g^{(\alpha+r)/\beta}, D_{i,j} = g^{r^*} \cdot H(a_{i,j})^{r''}, D'_{i,j} = g^{r''}$ for $0 \le i \le m, 1 \le j \le n_i, E_i = g^{(r+r^*)/(t\beta)}$, for $1 \le i \le m)$. Then the private key is returned to the adversary $\mathcal{A}$.

Note that attribute a_j in SK or SK′ may appear multiple times in $\mathbf{MK}^*$. The above private key derivation deals with this issue by randomly selecting t_i and $t_{i,j}$ from Z_p.

If the adversary $\mathcal{A}$ requests for a lower-level domain authority's private key or an end user's private key, it is noted that the master key MK_i of the domain authority DA_i can be obtained by querying $\mathrm{CreateDA}$ and $\mathrm{CreateUser}$ for some times ($\mathrm{CreateUser}$ should be queried for multiple times when there are multiple layers of domain authorities). Though MK_i may contain attributes that satisfy $\mathbb{A}^*$, only attributes in S are actually used in $\mathrm{CreateUser}$. It follows that $\mathcal{B}$ can answer the adversary's query by executing

the algorithm $\mathrm{CreateUser}$ using the attributes in S only, and returns the result to $\mathcal{A}$.

- **Challenge**. When $\mathcal{A}$ decides that Phase 1 is over, it outputs an access structure $\mathcal{T}$ and two messages $M_0, M_1 \in \mathbb{G}$, which it wishes to be challenged. $\mathcal{B}$ gives the two messages to CP-ABE challenger, and is given the challenge ciphertext $\mathbf{CT} = (\mathcal{T}, \tilde{C} = M_b \cdot e(g,g)^{\alpha s}, C = h^s, \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H(\mathbf{att}(y))^{q_y(0)})$.
  Then $\mathcal{B}$ computes the challenge ciphertext for $\mathcal{A}$ from $\mathbf{CT}$ as: $\mathbf{CT}^* = (\mathcal{T}, \widetilde{C} = M_b \cdot e(g,g)^{\alpha \cdot s}, C = h_1^s, \bar{C} = h_2^s, \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H(\mathbf{att}(y))^{q_y(0)}, \forall x \in X : \hat{C}_x = h_2^{q_x(0)})$. In $\mathbf{CT}^*$, $\bar{C}$, C_y, and C'_y are readily obtained from $\mathbf{CT}$. Note that $q_x(0)$ is a linear combination of s and other known values, which are determined by the public access structure. Thus $h_2^{q_x(0)}$ can be computed from h_2^s and other known values. Finally, the challenge ciphertext $\mathbf{CT}^*$ is returned to the adversary $\mathcal{A}$.
- **Phase 2**. $\mathcal{A}$ issues queries not issued in Phase 1. $\mathcal{B}$ responds as in Phase 1.
- **Guess**. Finally, $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$, and then $\mathcal{B}$ concludes its own game by outputting b'. According to the formal security model, the advantage of the adversary $\mathcal{B}$ against HASBE is

$$\mathrm{Adv}_{\mathcal{B}} = |\Pr[b = b'] - 1/2| = \mathrm{Adv}_{\mathcal{A}}.$$

  This means $\mathcal{B}$ has nonnegligible advantage against the CP-ABE scheme, which completes the proof of the theorem.

■

B. Discussion

In this subsection, we compare our scheme with the one proposed by Yu *et al.* [17] on security features in implementing access control for cloud computing.

1) Scalability: We extend ASBE with a hierarchical structure to effectively delegate the trusted authority's private attribute key generation operation to lower-level domain authorities. By doing so, the workload of the trusted root authority is shifted to lower-level domain authorities, which can provide attribute key generations for end users. Thus, this hierarchical structure achieves great scalability. Yu *et al.*'s scheme, however, only has one authority to deal with key generation, which is not scalable for large-scale cloud computing applications.
2) Flexibility: Compared with Yu *et al.*'s scheme, HASBE organizes user attributes into a recursive set structure and allows users to impose dynamic constraints on how those attributes may be combined to satisfy a policy. So HASBE can support compound attributes and multiple numerical assignments for a given attribute conveniently. As illustrated with the example key structure in Fig. 2 and access structure in Fig. 3, HASBE can enforce more complex access policies than Yu *et al.*'s scheme.
3) Fine-grained access control: Based on HASBE, our scheme can easily achieve fine-grained access control. A data owner can define and enforce expressive and flexible access policy for data files as the scheme in [17].
4) Efficient User Revocation: To deal with user revocation in cloud computing, we add an $\mathrm{expiration\_time}$ attribute to each user's key and employ multiple value assignments for this attribute. So we can update user's key by simply adding a new expiration value to the existing key. We just require a domain authority to maintain some state information of the user keys and avoid the need to generate and distribute new keys on a frequent basis, which makes our scheme more efficient than existing schemes.
5) Expressiveness: In HASBE, a user's key is associated with a set of attributes, so HASBE is conceptually closer to traditional access control methods such as Role-Based Access Control (RBAC) [18]. Thus, it is more natural to apply HASBE, instead of KP-ABE, to enforce access control.

VI. Performance Analysis and Implementation

In this section, we first analyze theoretic computation complexity of the proposed scheme in each operation. Then we implement an HASBE toolkit based on the cpabe toolkit developed for CP-ABE [18], and conduct a series of experiments to evaluate performance of our proposed scheme.

A. Performance Analysis

We analyze the computation complexity for each system operation in our scheme as follows.

System Setup. When the system is set up, the trusted authority selects a bilinear group and some random numbers. When $\mathbf{PK}$ and MK_0 are generated, there will be several exponentiation operations. So the computation complexity of *System Setup* is $O(1)$.

Top-Level Domain Authority Grant. This operation is performed by the trusted authority. The master key of a domain authority is in the form of $\mathrm{MK}_i = (\mathbb{A}, D, D_{i,j}, D'_{i,j} \text{ for } a_{i,j} \in \mathbb{A}, E_i \text{ for } A_i \in \mathbb{A})$, where $\mathbb{A}$ is the key structure associated with a new domain authority, A_i is the set of $\mathbb{A}$. Let N be the number of attributes in $\mathbb{A}$, and M be the number of sets in $\mathbb{A}$. Then the computation of MK_i consists of two exponentiations for each attribute in $\mathbb{A}$, and one exponentiations for every set in $\mathbb{A}$. The computation complexity of *Top-Level Domain Authority Grant* operation is $O(2N + M)$.

New User/Domain Authority Grant. In this operation, a new user or new domain authority is associated with an attribute set, which is the set of that of the upper level domain authority. The main computation overhead of this operation is rerandomizing the key. The computation complexity is $O(2N + M)$, where N is the number of attributes in the set of the new user or domain authority, and M is the number of sets in $\mathbb{A}$.

New File Creation. In this operation, the data owner needs to encrypt a data file using the symmetric key DEK and then encrypt DEK using HASBE. The complexity of encrypting the data file with DEK depends on the size of the data file and the underlying symmetric key encryption algorithm. Encrypting DEK with a tree access structure $\mathcal{T}$ consists of two exponentiations per leaf node in $\mathcal{T}$ and one exponentiation per *translating node* in $\mathcal{T}$. So the computation complexity of *New File Creation* is $O(2|Y| + |X|)$,

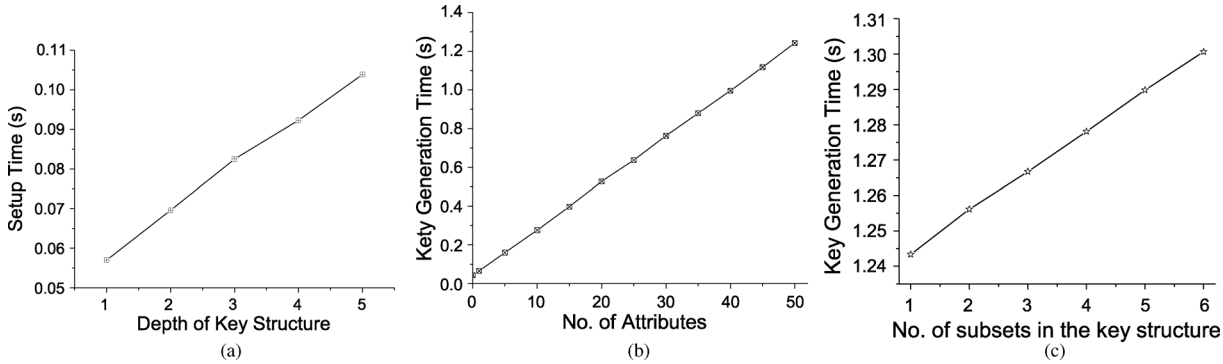

Fig. 6. Experiments on system setup and top-level domain authority grant. (a) Setup operation; (b) top-level domain authority grant (the number of subsets in the key structure is 1); (c) top-level domain authority grant (the total number of attributes in the key structure is 50).

TABLE I
COMPARISON OF COMPUTATION COMPLEXITY

Operation	HASBE	Yu's Scheme [17]
System Setup	$O(1)$	$O(\|Y\|)$
Top-level DA Grant	*O(2N+M)*	
User/DA Grant	*O(2N+M)*	$O(\|Y\|)$
File Creation	$O(2\|Y\| + \|X\|)$	$O(\|I\|)$
File Deletion	$O(1)$	$O(1)$
User Revocation	$O(1)$	$O(\|Y\|)$

where Y denotes the leaf nodes of $\mathcal{T}$ and X denotes the *translating nodes* of $\mathcal{T}$.

User Revocation. In this operation, a domain authority just maintains some state information of users' keys and assigns new value for expiration time to a user's key when updating it. When re-encrypting data files, the data owner just needs two exponentiations for ciphertext components associated with the $\mathrm{expiration\_time}$ attribute. So the computation complexity of this operation is $O(1)$.

File Access. In this operation, we discuss the decrypting operation of encrypted data files. A user first obtains DEKs with the $\mathrm{Decrypt}$ algorithm and then decrypt data files using DEKs. We will discuss the computation complexity of the $\mathrm{Decrypt}$ algorithm. The cost of decrypting a ciphertext varies depending on the key used for decryption. Even for a given key, the way to satisfy the associated access tree may be various. The $\mathrm{Decrypt}$ algorithm consists of two pairing operations for every leaf node used to satisfy the tree, one pairing for each *translating node* on the path from the leaf node used to the root and one exponentiation for each node on the path from the leaf node to the root. So the computation complexity varies depending on the access tree and key structure. It should be noted that the decryption is performed at the data consumers; hence, its computation complexity has little impact on the scalability of the overall system.

File Deletion. This operation is executed at the request of a data owner. If the cloud can verify the requestor is the owner of the file, the cloud deletes the data file. So the computation complexity is $O(1)$.

Computation complexity of each system operation is shown in Table I, in which N denotes the number of attributes in the key structure, I is the attribute set of the data file, Y is the set of leaf nodes of the access tree or policy tree, and X is the set of translating nodes of the policy tree.

B. Implementation

We have implemented a multilevel HASBE toolkit based on the cpabe toolkit (http://acsc.csl.sri.com/cpabe/) developed for CP-ABE [18] which uses the Pairing-Based Cryptography library (http://crypto.stanford.edu/pbc/). Then comprehensive experiments are conducted on a laptop with dual core 2.10-GHz CPU and 2-GB RAM, running Ubuntu 10.04. We make an analysis on the experimental data and give the statistical data. Similar to the cpabe toolkit, our toolkit also provides a number of command line tools as follows:

hasbe-setup: Generates a public key $\mathbf{PK}$ and a master key $\mathbf{MK}_0$.

hasbe-keygen: Given $\mathbf{PK}$ and $\mathbf{MK}_0$, generates a private key for a key structure. The key structure with depth 1 or 2 is supported.

hasbe-keydel: Given $\mathbf{PK}$ and $\mathbf{MK}_i$ of DA_i, delegates some parts of DA_i's private keys to a new user or DA_{i+1} in its domain. The delegated key is equivalent to generating private keys by the root authority.

hasbe-keyup: Given $\mathbf{PK}$, the private key, the new attribute and the subset, generates a new private key which contains the new attribute.

hasbe-enc: Given $\mathbf{PK}$, encrypts a file under an access tree policy specified in a policy language.

hasbe-dec: Given a private key, decrypts a file.

hasbe-rec: Given $\mathbf{PK}$, a private key and an encrypted file, re-encrypt the file. Note that the private key should be able to decrypt the encrypted file.

Fig. 6(a) shows the time required to setup the system for a different depth of key structure. Our scheme can be extended to support any depth of key structure. The cost of this operation increases linearly with the key structure depth, and the setup can be completed in constant time for a given depth. Except for this experiment, all other operations are tested with the key structure depth of 2.

Top-Level Domain Authority Grant is performed with the command line tool $\mathrm{hasbe-keygen}$. The cost is determined by the number of subsets and attributes in the key structure. When there is only one subset in the key structure, the cost grows linearly with the number of attributes as Fig. 6(b) shows. While the number of attributes in the key structure is fixed to be 50, the cost also increases linearly with the number of subsets as

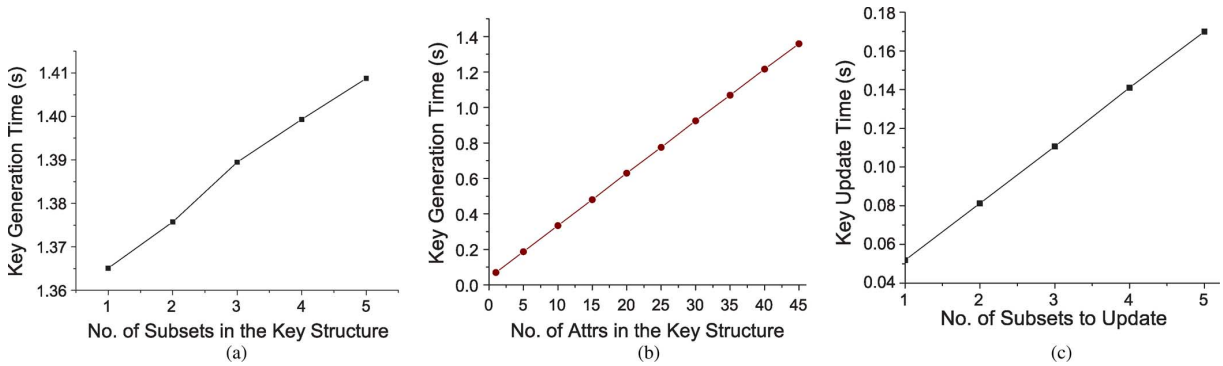

Fig. 7. Experiments on new user/domain authority grant and key update. (a) New user/domain authority grant (the total number of attributes in the master secret key of DA_i is 50 and the total number of attributes is 45); (b) new user/domain authority grant (the total number of attributes in the master secret key of DA_i is 50 and the number of subsets is 1); (c) key update (the total number of attributes in the original private key is 50).

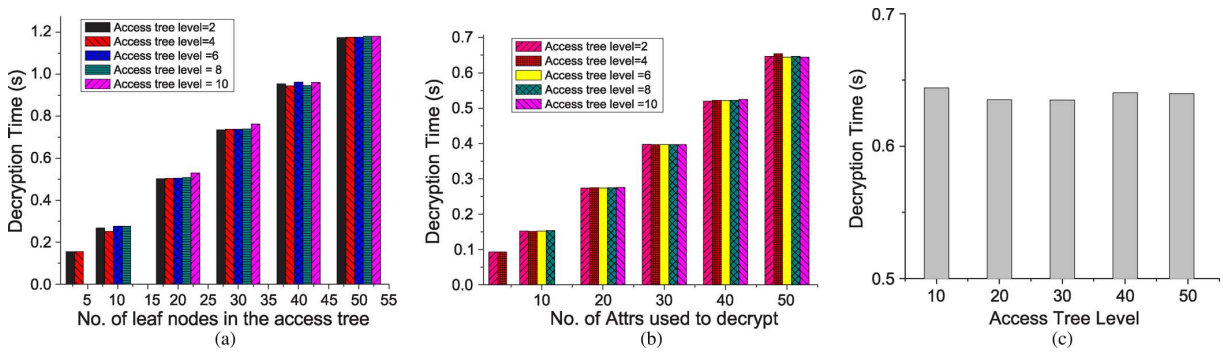

Fig. 8. Experiments on file creation and decryption. (a) Encryption/new file creation; (b) decryption/file access (there is 1 subset with 50 attributes in the private key); (c) decryption/file access (there is 1 subset with 50 attributes in the private key and the number of attributes used for decryption is 50).

shown in Fig. 6(c). Results of these two figures conform to the theoretic analysis.

With the command $\mathrm{hasbe-keydel}$, a domain authority DA_i can perform *New User/Domain Authority Grant* for a new user or another domain authority in his domain. The cost depends on the number of subsets and attributes to be delegated. Assume the domain authority DA_i has a private key with 50 attributes. When DA_i wants to delegate 45 of the attributes, the cost grows linearly with the number of subsets to be delegated as shown in Fig. 7(a). If DA_i delegates 1 of the subsets, the cost also increases linearly with the number of attributes in the subset as in Fig. 7(b).

User Revocation operation consists of two steps: *Key Update* and *Data Re-encryption*. *Key Update* is implemented with the command $\mathrm{hasbe-keyup}$. The root authority or domain authority can assign a new attribute to the user or domain authority. Adding a new attribute to one subset of private key can be done in constant time as the complexity is $O(1)$. If the new attribute needs to be assigned to several subsets, the cost is linear with the number of the subsets, as shown in Fig. 7(c).

Data Re-encryption is performed with the command $\mathrm{hasbe-rec}$. The data owner can re-encrypt the data file. For example, there is an encrypted file named $\mathrm{test.cpabe}$ which is encrypted with a policy a *and* b and the data owner re-encrypts it with the command $\mathrm{hasbe\text{-}recpub\_keyprv\_keytest.cpabe}$ c, then the new encrypted data file is associated with a policy a *and* b *and* c. When a user is revoked, the associated data file can be re-encrypted in this way, and the new attributes c can be assigned to valid user with command $\mathrm{hasbe-keyup}$. The cost of operation *Data Re-encryption* depends on the number of attributes on the access tree, which is same as the encryption operation, so we do not give the analysis here.

The data owner can use the command $\mathrm{hasbe-enc}$ to encrypt a file to create a new encrypted file. The time for this operation depends on the access tree structure. According to the number of leaf nodes and the level of the access tree policy, the time required to encrypt the file is shown in Fig. 8(a). We can see the cost is linear with the number of leaf nodes on the access tree and unrelated to the level of the access tree.

To access the file, decryption should be done with the command $\mathrm{hasbe-dec}$. The time of decryption is different depending on the access tree and key structure. Here we assume that there is just 1 subset with 50 attributes in the key structure associated with the private key. As shown in Fig. 8(b), the decryption time is proportional to the number of leaf nodes needed for decryption, and the level of the access tree has no impact on the decryption time.

In Fig. 8(c), assuming that the number of leaf nodes used for decryption is 50, we show the relationship between the access tree level and the time for decryption. We can see that the access tree level have no impact on the cost.

VII. Conclusion

In this paper, we introduced the HASBE scheme for realizing scalable, flexible, and fine-grained access control in cloud computing. The HASBE scheme seamlessly incorporates a hierarchical structure of system users by applying a delegation algorithm to ASBE. HASBE not only supports compound attributes due to flexible attribute set combinations, but also achieves efficient user revocation because of multiple value assignments

of attributes. We formally proved the security of HASBE based on the security of CP-ABE by Bethencourt *et al.*. Finally, we implemented the proposed scheme, and conducted comprehensive performance analysis and evaluation, which showed its efficiency and advantages over existing schemes.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] R. Buyya, C. ShinYeo, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Comput. Syst.*, vol. 25, pp. 599–616, 2009.

[2] Amazon Elastic Compute Cloud (Amazon EC2) [Online]. Available: http://aws.amazon.com/ec2/

[3] Amazon Web Services (AWS) [Online]. Available: https://s3.amazonaws.com/

[4] R. Martin, "IBM brings cloud computing to earth with massive new data centers," *InformationWeek* Aug. 2008 [Online]. Available: http://www.informationweek.com/news/hardware/data_centers/209901523

[5] Google App Engine [Online]. Available: http://code.google.com/appengine/

[6] K. Barlow and J. Lane, "Like technology from an advanced alien culture: Google apps for education at ASU," in *Proc. ACM SIGUCCS User Services Conf.*, Orlando, FL, 2007.

[7] B. Barbara, "Salesforce.com: Raising the level of networking," *Inf. Today*, vol. 27, pp. 45–45, 2010.

[8] J. Bell, Hosting Enterprise Data in the Cloud—Part 9: Investment Value Zetta, Tech. Rep., 2010.

[9] A. Ross, "Technical perspective: A chilly sense of security," *Commun. ACM*, vol. 52, pp. 90–90, 2009.

[10] D. E. Bell and L. J. LaPadula, Secure Computer Systems: Unified Exposition and Multics Interpretation The MITRE Corporation, Tech. Rep., 1976.

[11] K. J. Biba, Integrity Considerations for Secure Computer Sytems The MITRE Corporation, Tech. Rep., 1977.

[12] H. Harney, A. Colgrove, and P. D. McDaniel, "Principles of policy in secure groups," in *Proc. NDSS*, San Diego, CA, 2001.

[13] P. D. McDaniel and A. Prakash, "Methods and limitations of security policy reconciliation," in *Proc. IEEE Symp. Security and Privacy*, Berkeley, CA, 2002.

[14] T. Yu and M. Winslett, "A unified scheme for resource protection in automated trust negotiation," in *Proc. IEEE Symp. Security and Privacy*, Berkeley, CA, 2003.

[15] J. Li, N. Li, and W. H. Winsborough, "Automated trust negotiation using cryptographic credentials," in *Proc. ACM Conf. Computer and Communications Security (CCS)*, Alexandria, VA, 2005.

[16] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attibute-based encryption for fine-grained access control of encrypted data," in *Proc. ACM Conf. Computer and Communications Security (ACM CCS)*, Alexandria, VA, 2006.

[17] S. Yu, C. Wang, K. Ren, and W. Lou, "Achiving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. IEEE INFOCOM 2010*, 2010, pp. 534–542.

[18] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Security and Privacy*, Oakland, CA, 2007.

[19] R. Bobba, H. Khurana, and M. Prabhakaran, "Attribute-sets: A practically motivated enhancement to attribute-based encryption," in *Proc. ESORICS*, Saint Malo, France, 2009.

[20] A. Sahai and B. Waters, "Fuzzy identity based encryption," in *Proc. Acvances in Cryptology—Eurocrypt*, 2005, vol. 3494, LNCS, pp. 457–473.

[21] G. Wang, Q. Liu, and J. Wu, "Hierachical attibute-based encryption for fine-grained access control in cloud storage services," in *Proc. ACM Conf. Computer and Communications Security (ACM CCS)*, Chicago, IL, 2010.