International Refereed Journal of Engineering and Science (IRJES)
ISSN (Online) 2319-183X, (Print) 2319-1821
Volume 4, Issue 6 (June 2015), PP.186-198

Control Cloud Data Access Privilege and Anonymity with Fully Anonymous Attribute-Based Encryption

[1]Chirumarthi Venkanna,[2]G Shiva Krishna, [3]Naresh Badabath
[1,2,3] Computer Science Engineering Dept, Sree Dattha Institute Of Engineering & Science

ABSTRACT: Cloud computing is a revolutionary computing paradigm, which enables flexible, on-demand, and low-cost usage of computing resources, but the data is outsourced to some cloud servers, and various privacy concerns emerge from it. Various schemes based on the attribute-based encryption have been proposed to secure the cloud storage. However, most work focuses on the data contents privacy and the access control, while less attention is paid to the privilege control and the identity privacy. In this paper, we present a semi anonymous privilege control scheme Anony Control to address not only the data privacy, but also the user identity privacy in existing access control schemes. Anony Control decentralizes the central authority to limit the identity leakage and thus achieves semi anonymity. Besides, it also generalizes the file access control to the privilege control, by which privileges of all operations on the cloud data can be managed in a fine-grained manner. Subsequently, we present the AnonyControl-F, which fully prevents the identity leakage and achieve the full anonymity. Our security analysis shows that both AnonyControl and AnonyControl-F are secure under the decisional bilinear Diffie–Hellman assumption, and our performance evaluation exhibits the feasibility of our schemes.

Keywords: on-demand, encryption, Anony control,Diffie-Hellman

I.INTRODUCTION

Cloud computing is a new concept of computing technique, by which computer resources are provided dynamically via Internet. It attracts considerable attention and interest from both academia and industry. However, it also has at least three challenges that must be handled before applied to our real life. First of all, data confidentiality should be guaranteed. Whe n sensitive information is stored in cloud servers, which is out of users' control in most cases, risks would rise dramatically. The servers might illegally inspect users' data and access sensitive information. On the other hand, unauthorized users may also be able to intercept someone's data (e.g. server compromise). Secondly, personal information (defined by a user's attribu tes) is at risk because one's identity is authenticated according to his information. As people are becoming more concerned about their privacy these days, the privacy-preservability is very important. Preferably, any authority or server alone should not know any client's personal information. Last but not least, the cloud computing system should be resilient in the case of security breach in which some part of the system is compromised by attackers.

In fact, various techniques have been proposed and/or used to address the aforementioned problems. Identity-based encryption (IBE) was first introduced by Shamir in 1985 [1]. In the IBE, the sender of a message can specify an identity such that only a receiver with matching identity can decrypt it. This is different from Public-key Encryption, in that the encrypter does not need to issue extra key to decrypter for each ciphertext. In the IBE, the private key, which contains the identity of the holder, is distributed to every user only once when he joins the system.

Few years later, Sahai and Waters proposed a new type of IBE – Fuzzy Identity-Based Encryption [2], which is also known as Attribute-Based Encryption(ABE). In their work, an identity is viewed as a set of descriptive attributes. Different from the IBE, where the decrypter could decrypt the message if and only if his identity is exactly the same as what specifie d by the encrypter, this fuzzy IBE enables the decryption if there are `identity overlaps' exceeding a pre-set threshold between the one specified by encrypter and the one belongs to decrypter. However, this kind of threshold-based scheme was limited for designing more general system because the threshold based semantic cannot express a general condition.

Before long, more general tree-based ABE schemes, Key-Policy Attribute-Based Encryption (KP-ABE) [3] and Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [4], are proposed by Goyal *et al.* and Bethencourt *et al.* respec-tively to overcome the aforementioned drawback of fuzzy IBE. They look similar, but ciphertext and key structures are totally different, and the decision of encryption policy (who can or cannot decrypt the message) is made by different parties.

In the KP-ABE [3], a ciphertext is associated with a set of attributes, which partially represents the ciphertext's encryption policy. A private key is associated with a monotonic access structure like a tree, which describes this user's identity (e.g. IIT AND (Ph.D OR Master)). A user can decrypt the ciphertext if and only if the access tree in his private key is satisfied by the attributes in the ciphertext. However, th e encryption policy is described in the keys, so the encrypter does not have entire control over the encryption policy (who has access to the data and who does not). He has to trust that the key generators issue correct keys to correct users. Furthermore, when a re-encryption occurs, all of the users in the same system must have their private keys re-issued so as to gain access to the re-encrypted files, and this process cau ses considerable problems in implementation. On the other hand, those problems and overhead are all solved in the CP-ABE [4]. In the CP-ABE, ciphertexts are created with an access structure, which specifies the encryption policy, and priva keys are generated according to users' attributes. A user can decrypt the ciphertext if and only if his attributes in the private key satisfy the access tree specified in the ciphertext. By doing so, the encrypter holds the ultimate authority about the encryption policy. Also, the already issued private keys will never be modified unless the whole system crashes and the system's master key is lost.

In [5], [6], Chase introduced a multi-authority system, where each user has an ID and they can interact with each key generator (authority) using different pseudonyms. One user's different pseudonyms are tied to his private key, but key generators never know about the private keys, and thus they are not able to link multiple pseudonyms belonging to the same user. In fact they are even not able to distinguish the same user in different transactions. Also, the whole attributes set is divided into N disjoint sets and managed by N attributes authorities. That is, an attribute authority will only issue key components which it is in charge of. In this setting, even if an authority successfully guesses a user's ID, it knows only parts of the user's attributes, which are not enough to figure out the user's identity. However, the scheme proposed by Chase *et al.* [6] *considered* the basic threshold-based ABE, which is mentioned at the beginning of this section, and thus lacks expressibility in terms of encryption policy.

In addition, many similar literature works [7]–[11] have been published to create more advanced schemes where data needs to be securely and efficiently protected, which in turn served as the base of the research on security protocol in cloud computing environment [12]–[15]. However, much less effort is paid to protect users' privacy during those interactive protocols. Users' identities, which are described with their attributes, are generally opened to key generators, and the generators issue private keys according to their attributes. But it seems natural that users might want to keep their identities secret while they still get their private keys. Lewko's work [11] is the most similar one to ours in that they also tried to decentralize the central authority in the CP-ABE into multiple ones. They use a LSSS matrix as an access structure, but their scheme only converts the AND, OR gates to the LSSS matrix while we inherit the flexibility of the access tre e having threshold gates. In addition, our system tolerates the compromise attack towards attributes authorities, which is not covered in many existing works.

The main contributions of this paper are:

1) The proposed scheme is able to protect user's privacy against each single authority.

2) The proposed scheme is tolerant against authority com-promise, and compromising of up to (N − 2) authorities does not bring the whole system down.

3) We provide detailed analysis on security and performance to show feasibility of our scheme.

4) We first implement the real toolkit of multi-authority based encryption scheme.

II. PRELIMINARIES & RELATED WORK

A. Preliminaries

Let G_0 be a multiplicative cyclic group of prime order p and g be its generator. The bilinear map e is defined as follows: $e : G_0 \times G_0 \rightarrow G_T$, where G_T is the codomain of e. The bilinear map e has the following properties:

- Bilinearity: $\forall$ u, v $\in G_0$ and a, b $\in Z_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
- Symmetry: for all u, v $\in G_0$, e(u, v) = e(v, u).
- Non-*degeneracy*: e(g, g) =6 1.

Definition 1. *The Decisional Diffie-Hellman (DDH) problem in group* G_0 *of prime order* p *with generator* g *is defined as follows: on input* $g, g^a, g^b, g^c = g^{ab} \in G_0$, *where* a, b, c $\in Z_p$, *decide whether* c = ab *or* c *is a random element.*

Definition 2. *The Decisional Bilinear Diffie-Hellman (DBDH) problem in group* G_0 *of prime order* p *with generator* g *is defined as follows: on input* $g, g^a, g^b, g^c \in G_0$ *and* $e(g, g)^z = e(g, g)^{abc} \in G_T$, *where* a, b, c $\in Z_p$, *decide whether* z = abc *or* z *is a random element.*

The security of many ABE schemes (e.g. [4], [14], [16], [17]) and ours rely on the assumption that no probabilistic polynomial-time algorithms can solve the DDH and DBDH problem with non-negligible advantage. This assumption is reasonable since discrete logarithm problems in large number field are widely considered to be intractable ([18]–[21]), a nd therefore a is not deducible from g^a even if g is publicly

known.			
We also define the Lagrange coefficient	$\Delta_{i,S}$ for i ∈ Z_p and		
a set, S, of elements in Z_p: $\Delta_{i,S}$ (x) :=	x − j	, whi ch	
	j∈S,j=6i i−j		

Q

will be used in the polynomial interpolation in decryption algorithm to recover the secretly shared values. Additionally, a one-way hash function H : $\{0, 1\}^* \rightarrow G_0$ is also defined as a random oracle. This will map any attribute value to a random element in Z_p (same attribute value will be mapped to the same element).

B. CP-ABE

The idea of Attribute-Based Encryption (ABE) was first proposed by Sahai and Waters [2]. On the contrary to the traditional identity-based encryption, a user is able to decrypt a ciphertext if there is some match between his private key and ciphertext in the ABE. However, due to its lack of expressibility and generalization, it was later extended to the Key-Policy ABE by Goyal *et al.* [3] and the Ciphertext-Policy ABE by Bethencourt *et al.* [4]. Our scheme chooses CP-ABE as the base due to its advantages mentioned in the Section I.In the CP-ABE, the private key is distributed to users by a trusted central issuer only once. The keys are identified with a set of descriptive attributes, and the encrypter specifies an encryption policy using an access tree so that those with private keys which satisfy it can decrypt the ciphertext.

C. Privilege Trees T_p

In most of previous works [3], [4], [14], encryption policy is described with a tree called access tree. Each non-leaf node of the tree is a threshold gate, and each leaf node is described by an attribute. One access tree is required in every data file to define the encryption policy. In this paper, we extend existi ng schemes by generalizing the access tree to a privilege tree. The privilege in our scheme is defined as follows. A data file has several operations executable on itself, but some of them should be restricted only to authorized users. For example,

{*Read mine*, *Read all*, *Delete*, *Modify*, *Create*} is a privileges set of students' grades. Then, reading Alice's grades is allowed to her and her professors, but all other privileges should be authorized only to the professors, so we need to grant the " *Read mine*" to Alice and all other to the professors.

Every operation is associated with one privilege p, which is described by a privilege tree T_p. If a user's attributes satisfy T_p, he is granted the privilege p. By doing so, we not only control the file access but also control other executable operations , which makes the file controlling fine-grained and thus suitab le for cloud storage service. In our scheme, several trees are required in every data file to verify users' identity and to grant him a privilege accordingly. There are supposed to be r these kind of structures, which means there are r different privileges defined for the corre-sponding data file. The privilege 0 is defined as the privilege to read the file, and other privileges may be defined arbitrari ly (the m^{th} privilege does not necessarily have more powerful privilege than the n^{th} one when m > n). The tree is similar to the one defined in [4]. Given a tree, if num_x is the number of the node *x*'s children node and k_x is its threshold value $0 < k_x \leq num_x$, then node x is assigned a true value if at least k_x children nodes have been assigned true value. Specially, the node becomes an OR gate when $k_x = 1$ and an AND gate when $k_x = num_x$.

Several subsidiary functions are to be mentioned for conve-nience. We denote the parent of a node x by parent(x), and the attribute value of a leaf node x by att(x). Furthermore, the privilege tree T_p also defines the order between children of every node, and the numbers associated with node x, from 1 to num_x, are denoted by index(x).

	Table I
	Notati Ons F Or P Ri Vi Lege Trees

T_p	P^{th} Privilege Tree Representing The P^{th} Privilege
K_x	Threshold Value Of The Node X
Num_x	Number Of X's Child Nodes
Att(X)	Attribute Value Of The Node X, If It Is A Leaf Node
Index(X)	Index Of The X's Child Nodes
Parent(X)	Node X's Parent Node

D. Satisfying the Privilege Tree

If a user's attributes set S satisfies the privilege tree T_p or the node x, we define it as $T_p(S) = 1$ or $x(S) = 1$ respectively. $T_p(S)$ is calculated recursively as follows. If x is a leaf node, $x(S) = 1$ if and only if $att(x) \in S$. If x is a non-leaf node, $x(S) = 1$ only when at least k_x child nodes return 1. For the root node R_p of T_p, $T_p(S) = 1$ only if $R_p(S) = 1$.

Figure 1 shows an example of the privilege tree T_p for deleting the file. For an instance, if a user's attributes set is

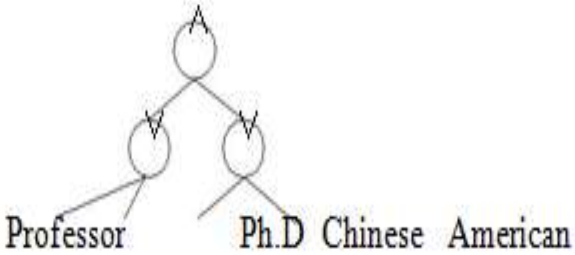


Fig. 1. An example of a privilege tree.

{Sex:Male, Age:23, Nationality:Chinese, University:Tsinghua University, Position:Ph.D Student, Religion:None}, he satisfies the tree, and thus achieving the corresponding privilege (here it means he can delete the file).

III. DEFINITIONS OF OUR SCHEME

A. System Model

In our system, there are four types of entities: N *Attribute Authorities* (denoted as A), *Cloud Server*, *Data Owners* and *Data Consumers*. A user can be a Data Owner and a Data Consumer simultaneously.

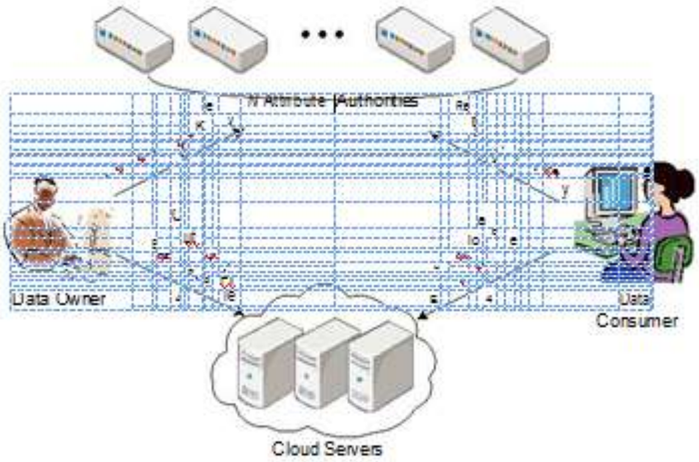


Fig. 2. Our system model

Authorities are assumed to have powerful computation abilities, who are supervised by government offices since keys act as IDs and partially contain users' PII (Personally Identifiable Information). The whole attribute set is divid ed into N disjoint sets and controlled by each authority. One practical method to divide the attributes set is to divide them by category (e.g., {**Sex**: Male, Female}, {**Nationality**: Korean, Chinese, Japanese}, {**University**: Tsinghua, Peking Univer-sity}, {**Position**: Professor, Ph.D Student, Master Student}). In this way, since each authority is aware of only one type of attribute, no useful information is leaked. The authorities jointly compute a system-wide public key, and individually compute their master keys at the initialization phase. The public key is used for all operations within the system, and the master keys are used by each attribute authority when he generates private keys for Data Consumers.

A Data Owner achieves public key from any one of the authorities, and he uses the public key to encrypt the data file before outsourcing it to the Cloud Servers. The Cloud Server, who is assumed to have adequate storage capacity, does nothing but store them.

Newly joined Data Consumers request private keys from all of the authorities, and they do not know which attributes are controlled by the authorities. On the other hand, authorities do not know which Data Consumers are interacting with them because each of them knows only a part of Data Consumers' attributes. When the Data Consumers request their private keys from the authorities, authorities jointly create corresponding private key and send it to them.

All Data Consumers are able to download any of those data files, but only those whose private keys satisfy the privileg e tree T_p can execute the operation associated with privilege p. When a user wants to execute a specific operation upon a data, he should satisfy the relevant privilege tree T_p and gets verified by the Cloud Server. The server is delegated to execute an operation p if and only if the user's privilege is verified through the privilege tree T_p.

B. Threats Model

We assume the Cloud Servers are untrusted, who behave properly in most of time but may collude with malicious Data Consumers or Data Owners to harvest others' file contents to gain illegal profits. But they are also assumed to gain legal benefit when users' requests are correctly processed, which means they will follow the protocol in general. In addition, even if the Cloud Server illegally modifies data files for sake of monetary benefits (e.g. deleting rarely accessed files to s ave the storage), whether the data is intact can be detected by the TPA technique introduced in [22].

The N authorities are assumed to be semi-honest. That is, they will follow our proposed protocol in general, but try to find out as much information as possible individually. More specifically, we assume they are interested in users' attrib utes to achieve the identities, but they will never collude with any user or authority to harvest file contents even if it is highly beneficial. This assumption is similar to many previo us researches on security issue in cloud computing (e.g. [14], [22]–[24]), and it is also reasonable since these authoriti es will be audited by government offices.

Data Consumers are untrustful since they are random users including attackers. They may collude with other Data Con-sumers to access what not allowed for them.

C. Design Goal

Our goal is to help Data Owners securely share their data with Data Consumers, where fine-grained privilege control is achievable, and to guarantee the confidentiality of Data Consumers' identity information by decomposing a center authority to multiple ones while preserving tolerance to com-promise attacks on the authorities. We assume the identity information is not disclosed by the underlying network. This can be achieved by employing anonymized protocols (e.g., [25]).

In the rest of this paper, A^u is used to denote the attributes set of a user u. A_k is used to denote the attribute authority k, and we also use a subscript k to denote the attributes set handled by Ak.

D. Definition of our Multi-Authority CP-ABE

Setup → PK, MK_k:

This algorithm takes nothing as input. Attributes authorities execute this algorithm to jointly compute a system-wide public

4

TABLE II
NOTATI ONS F OR SYS TEM CONS TRUCTI ON

Attribute Authorities	
k	index of an attribute authority
A_k	the k^{th} attribute authority
^{s}kj	secret parameter for compromise tolerance

Data Owner	
u	a user (either Data Owner or Data Consumer)
$_{A}$u	attributes set of user u
$_{A}{}^{T}$p	attributes set included in tree T_p
K_e	symmetric encryption/decryption key

parameter PK, authority-wide public parameter y_k and to individually compute master keys MK_k.
KeyGenerate(PK, MK_K, A^u, GID_u) → SK_u:

This algorithm enables a user to interact with every attribute authority, and obtains a private key SK_u corresponding to the input attribute set A^u and his global ID GID_u. From the authorities' perspective, the algorithm enables them to jointly generate a private key SK_u, using the public key PK and master keys MK_k, according to the input A^u and a pseudonym nym_{GIDu} , which is created according to the GID_u. Authorities are not able to derive a user's GID_u based on the pseudonym nym_{GIDu} .

Encrypt(PK, M , $\{T_p\}_{p\in\{0,\cdots,r-1\}}$) → (CT, VR):

This algorithm takes as input the public key PK, a message M , and a set of privilege trees $\{T_p\}_{p\in\{0,\cdots,r-1\}}$, where r is determined by the encrypter. It will encrypt the message M and returns a ciphertext CT and a verification set VR so that a user can execute specific operation on the ciphertext if and only if his attributes satisfy the corresponding privilege tree T_p. As we defined, T_0 stands for the privilege to read the file.
Decrypt(PK, SK_u, CT) → M or verification parameter:

This algorithm will be used at file controlling (e.g. reading , modification, deletion). It takes as input the public key PK, a ciphertext CT, and a private key SK_u, which has a set of attributes A^u and corresponds to its holder's GID_u. If the set A^u satisfies any tree in the set $\{T_p\}_{p\in\{0,\cdots,r-1\}}$, the algorithm returns a message M or a verification parameter. If the verification parameter is successfully verified by Clo ud Servers, who use VR to verify it, the operation request will be processed.
ReEncrypt(PK, CT, SK_u, $\{T_p{}'\}$) → CT', VR':

This algorithm is barely a composition of the decryption and the encryption algorithm. It takes as input the public key PK, a private key SK_u, a ciphertext CT, and a set of new privilege trees $\{T_p{}'\}_{p\in\{0,\cdots,r-1\}}$. If the set A^u in **SK**$_u$ satisfies T_0 (to obtain the original message M first) and T_k (privilege k is the one for re-encryption), the algorithm re-encrypts the original message M under new set of privilege trees and returns a new ciphertext CT' and a new verification set VR'

IV. OUR ANONYCONTROL SCHEME

A. Setup

At the system initialization phase, any one of the authorities chooses a bilinear group G_0 of prime order p with generator g and publishes it. Then, all authorities independently and
Then, every authority A_k randomly picks N − 1 integers $s_{kj} \in Z_p(j \in \{1, \cdots , N\}\backslash\{k\})$ and computes g^{skj} . Each g^{skj} is shared with each other authority A_j An authority A_k , after
receiving N − 1 pieces of g^{sjk}
secret parameter $x_k \in Z_p$ as follows:

$$x_k = \left(\prod_{j\in\{1,\cdots,N\}\backslash\{k\}} g^{s_{kj}}\right)/\left(\prod_{j\in\{1,\cdots,N\}\backslash\{k\}} g^{s_{jk}}\right) = g^{\left(\sum_{j\in\{1,\cdots,N\}\backslash\{k\}} s_{kj} - \sum_{j\in\{1,\cdots,N\}\backslash\{k\}} s_{jk}\right)}$$

Then, the master key for the authority A_k is $MK_k = \{v_k, x_k\}$, and public key of the whole system is published as $PK = \{G_0, g, Y = e(g, g)^{\overset{\mathbf{P}}{vk}}\}$.

Note that the time complexity of the setup computation is $O(N^2)$ since every authority computes $N - 1$ pieces of g^{skj} . However, this can be further reduced to $O(N)$ by applying the following simple trick. We first cluster the authorities int o C clusters, and exchanges the parameters within the cluster only. Then, the time complexity is reduced to $O(CN) = O(N)$ since C is a constant. Yet, this trick will also relax our compromise tolerance also. We will further discuss the trade-off between time complexity and compromise attack tolerance in the Section V.

B. *KeyGenerate(*PK, MK$_K$, A^u*)*

When a new user u with GID_u wants to join the system, he requests the private key from all of the authorities by following this process which is composed of two phases.

1) **Attribute Key Generation:** For any attribute $i \in A^u$, every A_k randomly picks $r_i \in Z_p$ to individually compute the partial private key

$$H(att(i))^{ri}, D_i' = g^{ri}$$

Then, all of the authorities randomly picks $d_k \in Z_p$ (d_k for A_k) and compute $x_k \cdot g^{vk} \cdot g^{dk}$ and share it with others.

Then, authorities merge the partial private keys by comput-ing the following: (this can be individually done by any one of the authorities)

$$D = \overset{\mathbf{Y}}{\ } \; xkg^{v}k\; g^{d}k = g^{\overset{\mathbf{P}}{v}k + \overset{\mathbf{P}}{d}k}$$

$$D_i = H(att(i))^{ri} \cdot \overset{\mathbf{Y}}{\ } g^{dk} = H(att(i))^{ri} \cdot g^{(\overset{\mathbf{P}}{\ } dk)}$$

At the end, the D, D_i's and D_i''s are sent to the user u.

2) Key Aggregation: User u, after receiving D_i's and D_i''s, aggregates the components as his private key:

$$SK_u = \{D, \forall i \in A^u : D_i = g^{(\overset{\mathbf{P}}{\ } dk)} \cdot H(att(i))^{ri}, D_i' = g^{ri}\}$$

C. *Encrypt*

Encryption must be done before Data Owners upload their data files to the Cloud Server. At first, he randomly selects a symmetric data encryption key K_e and encrypts the data file with it using any symmetric encryption scheme (e.g., 256-bit AES). Then, he determines a set of privilege trees $\{T_p\}_{p\in\{0,\cdots,r-1\}}$ and executes Encrypt(PK, K_e, $\{T_p\}$). For each T_p, the algorithm first chooses a polynomial q_x for each node x in it. For each node x, sets the degree d_x of the polynomial q_x as one less than the threshold value k_x. Starting from the root node R_p, the algorithm randomly picks $s_p \in Z_p$ and sets $q_{Rp}(0) := s_p$. Then, it chooses other numbers so that for any other node x, $q_x(0) = q_{parent(x)}(index(x))$ and randomly defines the q_x too.

Here, Shamir's secret sharing technique [26] is directly used to implement the threshold gate. Shamir's t-out of-n secret share scheme allows one to divide a secret to n shares, and the original secret can be recovered with t of them. So, in our tree, the node value of the gate is recovered if and only if at least k_x values of children nodes are recovered in recursive manner.

Finally, the ciphertext CT is created as

$$CT = h\{T_p\}_{p\in\{0,\cdots,r-1\}}, E_0 = K_e \cdot Y^{s0},$$

$$\{C_i = g^{qi\,(0)}, C_i' = H(att(i))^{qi\,(0)}\}_{i\in A}T_{p,\forall p\in\{0,\cdots,r-1\}}i$$

In the CT above, E_0 contains the symmetric key for decryption, and C_i's and C_i''s represent the

attribute values in the specified privilege trees.
Then, VR, which is disclosed only to the Cloud Server, is created for the purpose of privilege verification.
VR = h{E_p = Y ^{sp} }$_{p\in\{1,\cdots\ ,r-1\}}$i

Finally, Data Owner selects a unique ID for this encrypted data file and sends CT, VR and the encrypted file to the Cloud Server to share them with other Data Consumers.

ID	CT	Encrypted Data File

(a) seen from the data consumers' perspective.

ID	CT	VR	Encrypted Data File

(b) seen from the server's perspective.

Fig. 3. A data file stored on the cloud.

*D. Decrypt(*PK, SK$_u$, CT*)*

Every user within the system can download the ci-phertext from the Cloud Server, but he is able to exe-cute operations upon encrypted data only after he success-fully decrypts it. Firstly, we define a recursive algorithm DecryptN ode(CT, SK$_u$, x), where x stands for a node in the privilege tree T_p. If the node x is a leaf node, we let i be the attribute of the node x and define as follows. If i ∈ A^u,

DecryptN ode(CT, SK^u, x) = $^{e(Di\ ,Cx)}$

$\overline{e(D_i{}' ,C_x{}')}$

e($\quad$ (d_k

$\underline{g\ ^{\mathbf{P}\ d}k\ _{\cdot H(att(i))}{}^{r}i\ _{,g}{}^{qx\ (0)}\ )}$ $\quad$)$\cdot q_x(0)$

= $\quad e(g^ri\ ,H(att(i))^qx\ ^{(0)}\ )\ = e(g, g)\ ^{\mathbf{P}}$

If not, we define DecryptN ode(CT, SK$_u$, x) := ⊥.

If x is not a leaf node, the algorithm proceeds as fol-lows: For all nodes z that are children of x, it calls DecryptN ode(CT, SK$_u$, z) and stores the output as F_z . Let S_x be an arbitrary k_x-sized set of child nodes z such that F_z 6= ∅. If no such set exists then the node was not satisfied and the algorithm returns ⊥. Otherwise, compute

$S_x{}' = index(z) : z \in S_x$

$^{\triangle}d,s'\ \ (0)$ $\qquad$ $d = index(z)$

$^{\mathbf{Y}}Z$

x

$F_x =$ $\quad$ Fz $\quad$, where

z∈ S

= $\quad _{(e(g,\ g)}{}^{(\mathbf{P}\ d}k\ ^{)\cdot q}z\ ^{(0)}{}_{)}{}^{\triangle}d,S_x\quad (0)$

$\mathbf{Y}_z$

,

z∈ S

= $\quad _{(e(g,\ g)}{}^{(\mathbf{P}\ d}k\ ^{)\cdot q}parent(z)\ ^{(d)}{}_{)}{}^{\triangle}d,S_x\ ^{(0)}$

$\mathbf{Y}_z$

,

=z∈ S

$$\mathbf{Y}\ _{(e(g,\ g)}{}^{(\mathbf{P}\ d}k\ ^{)\cdot q}x^{(d)}{}_{)}{}^{\Delta}d,S_x \qquad {}_{,}(0)$$

$z \in S_z$

$$= e(g,\ g)^{(\ \overset{\mathbf{P}}{}\ dk\ )\cdot qx(0)}$$ (using polynomial interpolation)

The interpolation above recovers the parent node's value by calculating coefficients of the polynomial and evaluating t he p(0). We direct the readers to [26] for complete calculation. A user recursively calls this algorithm, starting from the root node R_p of the tree T_p, after downloading the file. If the tree is satisfied, which means he is granted the privilege p, then

$$\text{DecryptN ode}(CT,\ SK_u,\ R_p) = e(g,\ g)^{sp\quad dk}\ \mathbf{P}$$

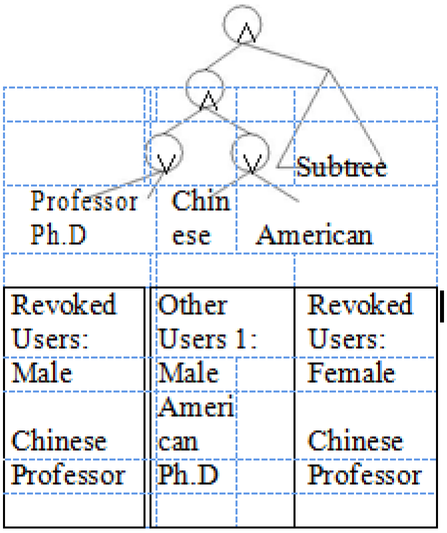
Finally, if the user is trying to read the file, the symmetric encryption key K_e can be recovered by:

$$\frac{\frac{E0}{e(g,D)}}{e(g,g)^{s}0\ ^{\mathbf{P}\ d}k} = \frac{\frac{K_e \cdot Y^{\ s0}}{e(g,g^{\mathbf{P}}\ d_k + ^{\mathbf{P}}\ v_k\ )}}{e(g,g)^{\mathbf{P}\ d}k} = K_e$$

Then, the data file can be decrypted by using it. Otherwise, he should be verified as an authorized user for the operation firs t. If the operation requires the j th privilege, the user recursively calls Decrypt(CT, SK_u, x) starting from the root node R_j of the tree T_j to get $e(g,\ g)^{sj\ dk}$ and further achieve $Y^{\ sj}$ with the same equation as above. The user sends it to the Cloud Server as well as the operation request. The Cloud Server checks whether $Y^{\ sj} = E_j$, and proceeds if they do equal each other. In fact, $Y^{\ sj}$ should be encrypted to avoid replay attack. This can be simply implemented by introducing any public key encryption protocol.

*E. ReEncrypt(*PK, CT, SK$_u$, $\{T_p{}'\}_{p\in\{0,\cdots,r-1\}}$*)*

In real applications in a cloud storage system, users might be revoked due to some reasons (e.g., resignation from a company). In this case, we need to re-encrypt the files to avoi d unauthorized access by revoked users, the users who satisfy certain properties for revocation (e.g., resignation). When they are revoked, they should not access the data files or execute other operations on them. An authorized user with the privilege to re-encrypt the associated file (note that this user might n ot be limited to the Data Owner) decrypts it first, and randomly selects another symmetric encryption key $K_e{}'$ to re-encrypt it. Then, he determines subtrees which forbid revoked users' access but still enables other unrelated users' one, and adds these subtrees into the original $\{T_p\}$ to gain new privilege set $\{T_p{}'\}$. Then, ReEncrypt(PK, CT, SK_u, $\{T_p{}'\}$) is executed to obtain new CT' and VT'.



Revoked Users:	Other Users 1:	Revoked Users:
Male	Male	Female
Chinese	American	Chinese
Professor	Ph.D	Professor

Fig. 4. An example of privilege tree after the re-encryption

Figure.4 shows a new tree $T_p^{'}$ after a re-encryption. As-suming that there are only three users within the system, who are described in Figure.4, and the `Revoked User' is revoked, the subtree may indicate the attribute condition as (Chinese ∨ Ph.D Student).

V. SECURITY ANALYSIS

A. User's Identity Information Confidentiality

The attributes, which contain a user's identity information, are separately controlled by different attribute authorities. Therefore, a user's attributes information is securely protected.

B. Trade-off between Tolerance and Complexity

In the proposed scheme, an authority A_k generates a set of random secret parameters $\{s_{kj}\}$ and shares it with other authorities, and the x_k is computed based on this parameters. Even if an adversary is able to compromise up to (N − 2) authorities, there are still two parameters kept unknown to the adversary. So, the adversary is not able to guess the valid ^{vk}, and he fails to construct a valid secret key. Hence, the scheme achieves compromise tolerance to up to (N − 2) authorities compromise.

But, if we reduce the time complexity of the setup phase by dividing authorities into several clusters having C authorities in each, attackers can compromise C −1 authorities in a cluster to create valid master keys of that cluster. Therefore, there is a trade-off between tolerance and complexity. However, since the number of authorities is typically not very huge, and the setup is one-time operation at the very beginning of the system setup, we recommend using the original setup algorithm whose complexity is $O(N^2)$.

Finally, note that the compromised authorities are able to issue valid attribute keys for which they are in charge of, so the ciphertexts whose privilege trees have only those attributes might be illegally decrypted if the attacker issue all possible attribute keys to himself. But, since the authorities are well protected servers, it is hard to compromise even one authority, and the probability of compromising enough authorities to illegally decrypt some ciphertext is very low.

C. Data Confidentiality against Collusion Attack

In order to access a plaintext, attackers must recover
s0 $= e(g, g)^{s0\ \mathbf{P}\ vk}$, which can be recovered only if the

attackers have enough attributes to satisfy the tree T_0. When two different keys' components are combined, the combined key cannot go through the polynomial interpolation in the decryption algorithm due to the randomization. Therefore, at least one key should be valid to satisfy the privilege tree.

Security Model for *AnonyControl* construction

W.L.O.G., we assume there is only one privilege tree T_0, which represents the privilege for reading, in the privilege set of CT. This assumption does not affect the security proof since we can easily extend the model to have several trees in CT. Next we show our scheme is secure against *chosen plaintext attacks* (CPA) if all probabilistic polynomial-time adversaries have negligible advantages in this game.

Init The adversary declares a privilege tree T_0^*, which he wants to be challenged.

Setup The challenger runs the Setup algorithm of our construction and publishes PK to the adversary.

Phase 1 The adversary queries for as many private keys, which correspond to attribute sets $A_1, \cdots, A_q$, as he wants, where none of these keys satisfy the T_0^* above.

Challenge The adversary submits two messages M_0 and M_1 of equal size to the challenger. The challenger flips a random binary coin b and encrypts M_b with T_0^*. The ciphertext CT^* is given to the adversary.

Phase 2 Phase 1 is repeated adaptively. Guess The adversary outputs a guess $b^{'}$ of b.

The advantage of an adversary A in this game is defined as $\Pr[b^{'} = b] - \frac{1}{2}$. We note that the model can easily be extended to handle chosen-ciphertext attacks by allowing for decryption queries in Phase 1 and Phase 2.

Here we prove that the security of our scheme in the security model above reduces to the intractability of the DBDH problem.

Definition 3. *Our scheme is secure against CPA if all polynomial-time adversaries have at most a negligible advan-tage in the above game.*

Theorem V.1. *If an adversary can break our scheme in the security model above, there exists at least one probabilistic polynomial-time algorithm can solve the DBDH problem, which is defined in the Section III, with a non-negligible advantage.*

Proof: Suppose there exists a probabilistic polynomial-time adversary A can attack our scheme in the security model above with advantage ϱ. We prove that the following DBDH game can be solved with advantage $\frac{\varrho}{2}$.

Let e : $G_0 \times G_0 \rightarrow G_T$ be a bilinear map, where G_0 is a multiplicative cyclic group of prime order p and g is its generator. First the DBDH challenger flips a binary coin μ, and he sets (g, A, B, C, Z) := $(g, g^a, g^b, g^c, e(g, g)^{abc})$ if μ = 0; otherwise he sets (g, A, B, C, Z) := $(g, g^a, g^b, g^c, e(g, g)^z)$,
where a, b, c, z ∈ Z_p are randomly picked. The challenger then gives the simulator hg, A, B, C, Zi = hg, g^a, g^b, g^c, Zi. The simulator sim then plays the role of a challenger in the following DBDH game.

Init The adversary A creates a T_0^* which he wants to be challenged (Nodes inside the tree should be defined by him). Setup sim sets the parameter Y := e(A, B) = $e(g, g)^{ab}$ and gives this public parameter to A.
Phase 1 A queries for as many private keys, which correspond to attribute sets $A_1, \cdots, A_q$, as he wants, where none of them satisfy the T_0^*. sim, after receiving the key queries, computes the components in private keys to respond
the A's requests. For all attributes i ∈ A^u, he randomly picks $r_i \in Z_p$, and computes $D_i := A \cdot H(att(i))^{ri}$, $D_i' := g^{ri}$ .Then, sim returns the created private key to A.

Challenge The adversary A submits two challenge messages m_0 and m_1 to the challenger. The challenger flips a binary coin γ, and returns the following ciphertext to A.

$$CT^* = hT_0^*, E_0 = m_\gamma \cdot Z,$$

$$\{C_i = g^{q_i(0)}, C_i' = H(att(i))^{q_i(0)}\}_{i \in A_0^{T^*}}$$

If μ = 0, Z = $e(g, g)^{abc}$. If we let ab = v_k k ∈ {1, 2, · · ·P , N } and c = s_0
(this is possible because v , k 0 and s are all randomly chosen from Z_p), we have Z = $e(g, g)^{abc}$ =
$(e(g, g)^{ab})^c = Y^{s0}$.

Therefore, CT^* is a valid ciphertext of the message m_γ . Otherwise, if μ = 1, Z = $e(g, g)^z$. Then, we have $E_0 = m_\gamma \cdot e(g, g)^z$. Since z ∈ Z_p is a random element, E_0 is a random element in G_T from A's perspective, therefore CT^* contains no information about m_γ .
Phase 2 Repeat Phase 1 adaptively.

Guess A submits a guess γ' of γ. If $\gamma' = \gamma$, sim outputs $\mu' = 0$, indicating that it was given a valid DBDH-tuple (g, A, S, Z), otherwise it outputs $\mu' = 1$, indicating that he was given a random 5-element tuple (g, A, B, C, Z).

As shown in the construction of the game, the simulator sim computes the public parameter and the private key in the same way as our scheme. When μ = 1, the adversary A learns no information about γ, so we have $Pr[\gamma =6 \gamma'|\mu = 1] = \frac{1}{2}$. Since the challenger guesses $\mu' = 1$ when $\gamma = \gamma'$, we have $Pr[\mu' = \mu|\mu = 1] = \frac{1}{2}$. If μ = 0, the adversary A gets a valid ciphertext of m_γ . A's advantage in this situation is ϱ by definition, so we have Pr $[\gamma = \gamma'|\mu = 0] = \frac{1}{2} + \varrho$. Since the challenger guesses $\mu' = 0$ when $\gamma = \gamma'$, we have $Pr[\mu' = \mu|\mu = 0] = \frac{1}{2} + \varrho$. The overall advantage in this
DBDH game is $\frac{1}{2} Pr[\mu' = \mu|\mu = 0] + \frac{1}{2} Pr[\mu' = \mu|\mu = 1] - \frac{1}{2}$
$= \frac{1}{2} \cdot (\frac{1}{2} + \varrho) + \frac{1}{2} \cdot \frac{1}{2} - \frac{1}{2} = \frac{\varrho}{2}$.

To conclude, as proved above, the advantage for a polynomial-time adversary in the DBDH game is $\frac{\varrho}{2}$ if the advantage for a polynomial-time adversary in our security model is ϱ. Therefore, if an adversary can break our scheme in our security model, which indicates ϱ is a non-negligible advantage, a polynomial-time adversary's advantage, which is $\frac{\varrho}{2}$, in solving the DBDH problem is also non-negligible. ■

Since our scheme relies on the assumption that no proba-bilistic polynomial algorithm can solve the DBDH problem

TABLE III Com P Lexi Ty Com Pari S On (P Er Authori Ty)

Process	Yu *et al.* [12]	Chase *et al.* [6]	Ours
Setup	O(I)	O(1)	O(1)
Key Generation	O(X)	O(N + I)	O(N + I)
Encryption	O(I)	O(I)	O(X · K)
Decryption	O(MAX(X, I))	O(N · I)	O(X)
User Revocation	O(I)		O(X · K)

with non-negligible advantage, it can be deduced that no adversary can break our scheme in our security model.

VI. PERFORMANCE ANALYSIS

In this section, we denote N as the number of attribute authorities, I as the size of the entire attribute set and X as the number of nodes in a tree T_p.

A. Setup

When the system is setup, $^{\mathbf{Q}}$ Y_k is computed by any one of the authorities and sent to others, whose complexity is O(N). Then, secret parameters x_k's are calculated within the clusters. The complexity of that calculation is O(C^2 · $^{N}{}_{C}$) = O(C · N), but C is a constant number, so O(C · N) = O(N). Therefore, the total complexity is O(N). However, since we have N authorities per system, the complexity per authority is O(1).

B. Key Generation

In the Attribute Key Generation, $g^{\mathbf{P}\ vj}$ is computed by N authorities, and D_i = $H(att(i))^{ri}$ · $g^{\mathbf{P}}$ v_j is computed for I times by one attribute authority. Therefore, the total complexity of Attribute Key Generation is O(N ^{2} + I · N). In the Aggregation of Two Keys, a user aggregates the I components, thus the computation complexity of this operation is O(I). So, the complexity per authority is O(N + I).

C. Encryption

At every non-leaf node, a polynomial is chosen and $k_x - 1$ numbers are to be found to determine the polynomial, where k_x is the threshold value. Therefore, denoting the average threshold value to be K,the computation complexity of this process is O(X · K).

D. Decryption

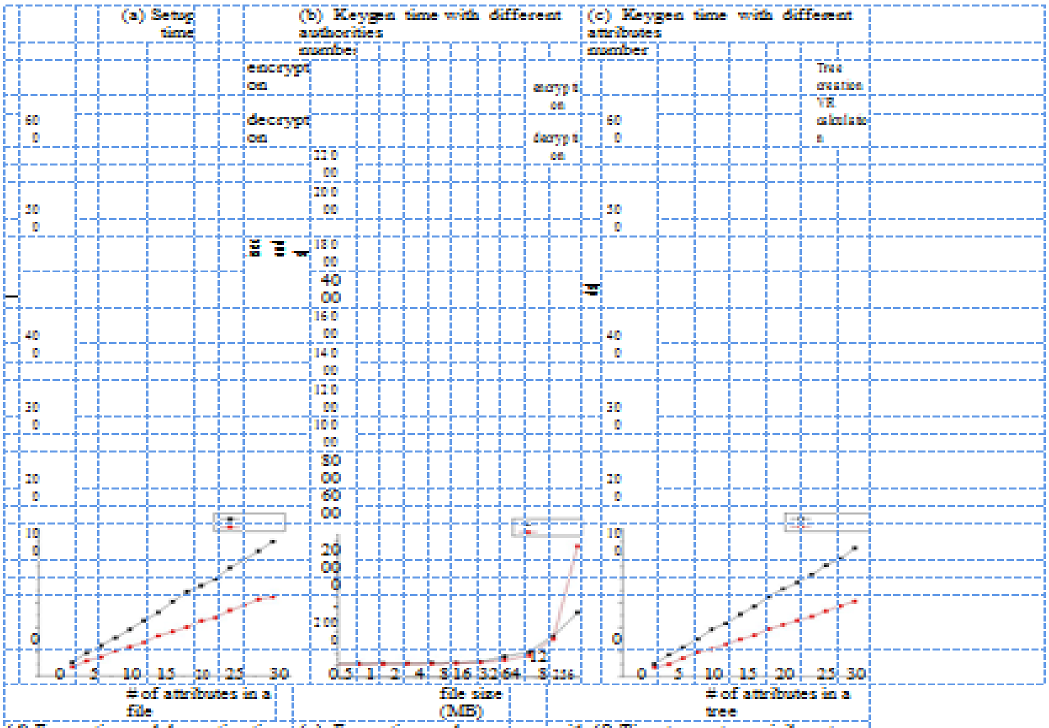
DecryptN ode is a recursive algorithm, and it is executed exactly once at every nodes in a Breadth-First-Search manner, therefore the computation complexity of this process is O(X).

Implementation

In this section, we give the experimental result of our scheme, which is conducted on the prototype of our scheme. To the best of our knowledge, this is the first implementation of a multi-authority attribute based encryption scheme. Our prototype system provides five command line tools. anonyabe-setup : Jointly generates a public key and N master keys.

VIII. CONCLUSION

This paper proposed an anonymous attribute-based privilege control scheme *AnonyControl* to address the user privacy problem in a cloud storage server. Using multiple authorities in the cloud computing system, our proposed scheme achieves not only fine-grained privilege control, but also anonymity while conducting privilege control based on users' identity information. More importantly, our system can tolerate up to N − 2 authority compromise, which is highly preferable especially in Internet-based cloud computing environment. Furthermore, although the data contents are fully outsourced to Cloud Servers, the Cloud Servers cannot read the contents unless their private keys satisfy the privilege tree T_0. We also conducted detailed security and performance analysis which shows that *AnonyControl* is both secure and efficient for cloud storage system.



(d) Encryption and decryption time with dif- ferent attributes number. File size is 100KB

(e) Encryption and decryption different file size. 20 attributes in

T_2

with (f) Time to create a privilege tree and de- crypt a verification parameter from it

Fig. 5. Experiment result on our implemented prototype system.

REFERENCES

[1] A. Shamir, “Identity-based cryptosystems and signatur e schemes,” in Advances in cryptology, Springer, 1985, pp. 47–53.

[2] A. Sahai and B. Waters, “Fuzzy identity-based encryptio n,” Advances in Cryptology–EUROCRYPT 2005 , pp. 557–557, 2005.

[3] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute- based encryp-tion for fine-grained access control of encrypted data,” in Proceedings of the 13th ACM conference on Computer and communications security, 2006, pp. 89–98.

[4] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-po licy attribute-based encryption,” in IEEE Symposium on Security and Privacy, 2007, pp. 321–334.

[5] M. Chase, “Multi-authority attribute based encryption ,” Theory of Cryp-tography, pp. 515–534, 2007.

[6] M. Chase and S. Chow, “Improving privacy and security in m ulti-authority attribute-based encryption,” in Proceedings of the 16th ACM conference on Computer and communications security, 2009, pp. 121– 130.

[7] J. Horwitz and B. Lynn, “Toward hierarchical identity-b ased encryption,” in Advances in Cryptology–EUROCRYPT 2002 , Springer, 2002, pp. 466–481.

[8] A. Boldyreva, V. Goyal, and V. Kumar, “Identity-based en cryption with efficient revocation,” in Proceedings of the 15th ACM conference on Computer and communications security, 2008, pp. 417–426.

[9] T. Nishide, K. Yoneyama, and K. Ohta, “Attribute-based e ncryption with partially hidden encryptor-specified access structur es,” in Applied Cryptography and Network Security, Springer, 2008, pp. 111–129.

[10] R. Ostrovsky, A. Sahai, and B. Waters, “Attribute-base d encryption with non-monotonic access structures,” in Proceedings of the 14th ACM conference on Computer and communications security, 2007, pp. 195– 203.

[11] A. Lewko and B. Waters, “Decentralizing attribute-bas ed encryption,” Advances in Cryptology–EUROCRYPT 2011 , pp. 568–588, 2011.