PUBLISHING YEAR 2012

Privacy Preserving Cloud Data Access With Multi-Authorities

Taeho Jung†, Xiang-Yang Li† and Zhiguo Wan‡
†Department of Computer Science, Illinois Institute of Technology, Chicago, IL, 60616
‡School of Software, Tsinghua University, Beijing, 100084

***Abstract*—Cloud computing is a revolutionary computing paradigm which enables flexible, on-demand and low-cost usage of computing resources. However, those advantages, ironically, are the causes of security and privacy problems, which emerge because the data owned by different users are stored in some cloud servers instead of under their own control. To deal with security problems, various schemes based on the Attribute-Based Encryption have been proposed recently. However, the privacy problem of cloud computing is yet to be solved. This paper presents an anonymous privilege control scheme *AnonyControl* to address the user and data privacy problem in a cloud. By using multiple authorities in cloud computing system, our proposed scheme achieves anonymous cloud data access, fine-grained privilege control. Our security and performance analysis shows that *AnonyControl* is both secure and efficient for cloud computing environment.**

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. When 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 attributes) 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. It would be preferable if the system is still secure when most of trusted authorities or servers are compromised.

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 systtem.

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 specified 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.* respectively to overcome the aforementioned drawback of fuzzy IBE. They look similar, but ciphertext and key structures are totally different, and thus the decision of encryption policy (who can decrypt the ciphertext and who can not) 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, the 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 modified so as to gain access to the re-encrypted files, and this process causes 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 private 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.

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, much similar research [7]–[10] has been conducted 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 [11]–[14]. For example, Yu *et al.* [11] proposed a scheme to achieve secure, scalable and fine-grained access control in cloud computing, using numerous techniques such as proxy re-encryption [15] and KP-ABE [3]. But their system incurs huge overhead every time a user is revoked, which is one of the disadvantages of KP-ABE. 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. The main challenge here is that key generators have to issue keys to corresponding users while they are forbidden to collect personal information of key recipients.

We introduce the first anonymous ciphertext-policy attribute-based encryption scheme based on a multi-authority model for cloud computing. The main contributions of this paper are: We propose a new ciphertext-policy attribute-based encryption scheme with multiple authorities, which can deliver fine-grained data control for cloud computing. The proposed scheme is able to protect user's privacy against each single authority. The proposed scheme is tolerant against authority compromise, and compromising of up to $(N-2)$ authorities does not bring the whole system down. We also provide detailed discussion on security and performance analysis to show effectiveness of the proposed scheme.

II. Preliminaries & Related Work

A. Preliminaries

Let $\mathbb{G}_0$ be a multiplicative cyclic group of prime order p and g be its generator. The bilinear map e is defined as follows: $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_T$, where $\mathbb{G}_T$ is the codomain of e. The bilinear map e has the following properties:

- *Bilinearity*: $\forall u, v \in \mathbb{G}_0$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
- *Symmetry*: for all $u, v \in \mathbb{G}_0$, $e(u, v) = e(v, u)$.
- *Non-degeneracy*: $e(g, g) \neq 1$.

Definition 1. *The Decisional Diffie-Hellman (DDH) problem in group* $\mathbb{G}_0$ *of prime order* p *with generator* g *is defined as follows: on input* $g, g^a, g^b, g^c = g^{ab} \in \mathbb{G}_0$, *where* $a, b, c \in \mathbb{Z}_p$, *decide whether* $c = ab$ *or* c *is a random element.*

Definition 2. *The Decisional Bilinear Diffie-Hellman (DBDH) problem in group* $\mathbb{G}_0$ *of prime order* p *with generator* g *is defined as follows: on input* $g, g^a, g^b, g^c \in \mathbb{G}_0$ *and* $e(g, g)^z = e(g, g)^{abc} \in \mathbb{G}_T$, *where* $a, b, c \in \mathbb{Z}_p$, *decide whether* $z = abc$ *or* z *is a random element.*

The security of many ABE schemes (e.g. [4], [13], [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.

We also define the Lagrange coefficient $\triangle_{i,S}$ for $i \in \mathbb{Z}_p$ and a set, S, of elements in $\mathbb{Z}_p$:

$$\triangle_{i,S}(x) := \prod_{j \in S, j \neq i} \frac{x - j}{i - j}, \tag{1}$$

which will be used in polynomial interpolation in decryption algorithm. Additionally, a hash function $H : \{0, 1\}^* \to \mathbb{G}_0$ is also defined as a random oracle. This will map any attribute value to a random element in $\mathbb{Z}_p$.

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], [13], 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 existing schemes by generalizing the access tree to privilege tree. The privilege in our scheme is defined as follows. There are several possible operations that can be executed on a data file, but some of the operations should be restricted only to authorized users. For example, {*Read_mine, Read_all, Delete, Modify, Write, Copy, Move*} is a privilege set of students' grades. In this example, reading Alice's grades is allowed to her and her professors, but modification or creation of her grades should be authorized only to the professors, so we need to grant the privilege "Read_mine" to Alice and all of the others to the professors.

Every operation is associated with one privilege (say p), which is described by a privilege tree (say T_p). If a user's attributes satisfy the privilege tree 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 suitable 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 corresponding data file. The privilege 0 is defined to be the privilege to read the file in our scheme, and the other privileges may be defined arbitrarily (the m^{th} privilege is not necessarily higher than the n^{th} one even if $m > n$). 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 convenience. 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
NOTATIONS FOR PRIVILEGE 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 {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).

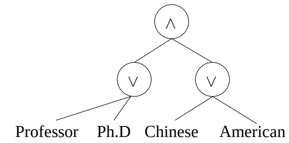


Fig. 1. An example of a privilege tree.

III. DEFINITIONS OF OUR SCHEME

A. System Model

In our system, the cloud computing system consists of four types of entities: N *Attribute Authorities* (denoted as $\mathcal{A}$), *Cloud Server*, *Data Owners* and *Data Consumers*. A user can be a Data Owner and a Data Consumer simultaneously.

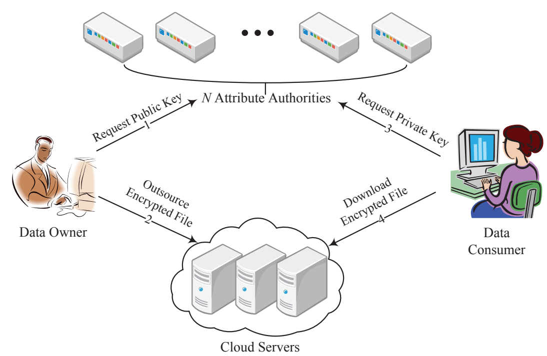


Fig. 2. Our system model

Authorities are assumed to have powerful computation abilities. The whole attribute set is divided 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 University}, {**Position**: Professor, Ph.D Student, Master Student}). 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 privilege 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 profits paid by users, when users' operation requests are correctly processed, which means the Cloud Servers 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 save the storage space), whether the data is intact can easily be detected by the Third Party Auditing technique introduced in [18].

The N authorities are assumed to be semi-honest. That is, attribute authorities 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' attributes to figure out their identities, but they will never collude with any user or authority for the purpose of harvesting file contents even if it is highly beneficial. This assumption is similar to many previous researches on security issue in cloud computing (e.g. [13], [18]–[20]).

Data Consumers are assumed to be malicious and untrustful. They may collude with other Data Consumers to access what they are not allowed to access.

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 confidentiality of Data Consumers' attributes information by decomposing a center authority to multiple authorities.

In the rest of this paper, $\mathbb{A}^u$ is used to denote the attribute set of a user u. $\mathcal{A}_k$ is used to denote the attribute authority k, and we also use subscript k to denote the attribute set handled by the $\mathcal{A}_k$.

D. Definition of out Multi-Authority CP-ABE

Setup $\rightarrow \mathbf{PK}, \mathbf{MK}_k$:
The setup algorithm takes nothing as input. Attributes authorities execute this algorithm to jointly compute a system-wide public parameter $\mathbf{PK}$, authority-wide public parameter y_k and to individually compute master keys $\mathbf{MK}_k$.

KeyGenerate($\mathbf{PK}$, $\mathbf{MK_k}$, $\mathbb{A}^u$, $\mathbf{GID}_u$) $\rightarrow \mathbf{SK}_u$:
The Key Generation algorithm enables a user to interact with every attribute authority, and obtains a private key $\mathbf{SK}_u$ corresponding to the input attribute set $\mathbb{A}^u$ and his global ID $\mathbf{GID}_u$. From the authorities' perspective, the algorithm enables them to jointly generate a private key $\mathbf{SK}_u$, using the public key $\mathbf{PK}$ and master keys $\mathbf{MK}_k$, according to the input $\mathbb{A}^u$ and a pseudonym $nym_{\mathbf{GID}_u}$, which is created according to the $\mathbf{GID}_u$. Authorities are not able to derive a user's $\mathbf{GID}_u$ based on the pseudonym $nym_{\mathbf{GID}_u}$.

TABLE II
NOTATIONS FOR SYSTEM CONSTRUCTION

Attribute Authorities	
k	index of an attribute authority
$\mathcal{A}_k$	the k^{th} attribute authority
s_{kj}	secret parameter for compromise tolerance
Data Owner	
u	a user (either Data Owner or Data Consumer)
$\mathbb{A}^u$	attributes set of user u
$\mathbb{A}^{T_p}$	attributes set included in tree T_p
K_e	symmetric encryption/decryption key

Encrypt($\mathbf{PK}$, M, $\{T_p\}_{p\in\{0,\cdots,r-1\}}$) $\rightarrow (\mathbf{CT}, \mathbf{VR})$:
The encryption algorithm takes as input the public key $\mathbf{PK}$, 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 $\mathbf{CT}$ and a verification set $\mathbf{VR}$ 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($\mathbf{PK}$, $\mathbf{SK}_u$, $\mathbf{CT}$) $\rightarrow M$ or verification parameter:
The decryption algorithm will be used at data file controlling (e.g. reading, modification, deletion). It takes as input the public key $\mathbf{PK}$, a ciphertext $\mathbf{CT}$, and a private key $\mathbf{SK}_u$, which has a set of attributes $\mathbb{A}^u$ and corresponds to its holder's $\mathbf{GID}_u$. If the set $\mathbb{A}^u$ 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 Cloud Servers, who use $\mathbf{VR}$ to verify it, the operation request will be processed.

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

IV. OUR ANONYCONTROL SCHEME

A. Setup

At the system initialization phase, any one of the authorities chooses a bilinear group $\mathbb{G}_0$ of prime order p with generator g. Then all authorities independently and randomly picks $v_k \in$

$\mathbb{Z}_p$ and send $Y_k = e(g,g)^{v_k}$ to all other authorities. Authorities individually compute $Y := \prod_{k \in \mathcal{A}} Y_k = e(g,g)^{\sum_{k\in\mathcal{A}} v_k}$ and send it to others.

Then, authorities are divided into several clusters. Within a cluster of C authorities (indexed as $1,\ 2,\ \cdots,\ C$), every authority $\mathcal{A}_k$ randomly picks $C-1$ integers $s_{kj} \in \mathbb{Z}_p (j \in \{1,\cdots,C\}\backslash\{k\})$ and computes $g^{s_{kj}}$. These are shared with each other authority $\mathcal{A}_j$. An authority $\mathcal{A}_k$, after receiving $C-1$ pieces of s_{jk} generated by $\mathcal{A}_j$, computes its secret parameter $x_k \in \mathbb{Z}_p$ as follows:

$$
\begin{aligned}
x_k &= (g^{\sum_{j\in\{1,\cdots,C\}\backslash\{k\}} s_{kj}})/(\prod_{j\in\{1,\cdots,C\}\backslash\{k\}} g^{s_{jk}}) \\
&= g^{(\sum_{j\in\{1,\cdots,C\}\backslash\{k\}} s_{kj} - \sum_{j\in\{1,\cdots,C\}\backslash\{k\}} s_{jk})}
\end{aligned}
$$

Then the master key for the authority $\mathcal{A}_k$ is $\mathbf{MK}_k = \{v_k, x_k\}$, and public key of the whole system is published as $\mathbf{PK} = \{\mathbb{G}_0, g, Y = e(g,g)^{\sum v_i}\}$.

It is easy to see that these randomly produced integers satisfying that $\prod_{k\in\mathcal{A}} x_k = 1 \mod p$. This is an important property for our scheme presented in the next section.

B. KeyGenerate($\mathbf{PK}$, $\mathbf{MK_k}$, $\mathbb{A}^u$)

When a new user u with $\mathbf{GID}_u$ wants to join the system, he requests the private key from all of the authorities by following this process.

1) Attribute Key Generation: For any attribute $i \in \mathbb{A}^u$, every $\mathcal{A}_k$ randomly picks $r_i \in \mathbb{Z}_p$ to individually compute the partial private key

$$x_k \cdot g^{v_k}, H(att(i))^{r_i}, D_i' = g^{r_i}$$

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

$$
\begin{aligned}
D_i &= H(att(i))^{r_i} \cdot \prod (x_k \cdot g^{v_k}) \\
&= H(att(i))^{r_i} \cdot g^{(\sum v_k)}.
\end{aligned}
$$

At the end, the 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:

$$\mathbf{SK}_u = \{\forall i \in \mathbb{A}^u : D_i = g^{(\sum v_k)} \cdot H(att(i))^{r_i}, D_i' = g^{r_i}\}$$

C. Encrypt($\mathbf{PK}$, M, $\{T_p\}_{p\in\{0,\cdots,r-1\}}$)

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. Then, he determines a set of privilege trees $\{T_p\}_{p\in\{0,\cdots,r-1\}}$ and executes $Encrypt(\mathbf{PK}, K_e, \{T_p\})$. For each T_p, the algorithm first chooses a polynomial q_x for each node x in it. For each node x, set the degree d_x of the polynomial q_x to be one less than the threshold value k_x. Starting from the root node R_p, the algorithm randomly picks $s_p \in \mathbb{Z}_p$ and sets $q_{R_p}(0) := s_p$. 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 [21] is directly used to implement the threshold node. 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.

Finally, the ciphertext $\mathbf{CT}$ is created as

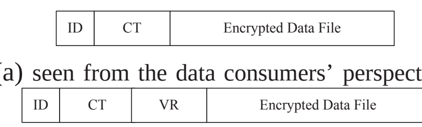
$$
\begin{aligned}
\mathbf{CT} = \langle &\{T_p\}_{p\in\{0,\cdots,r-1\}}, E_0 = K_e \cdot Y^{s_0}, \\
&\{C_i = g^{q_i(0)}, C_i' = H(att(i))^{q_i(0)}\}_{i\in\mathbb{A}^{T_p}, \forall p\in\{0,\cdots,r-1\}}\rangle
\end{aligned}
$$

In the $\mathbf{CT}$ 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, $\mathbf{VR}$, which is disclosed only to the Cloud Server, is created for the purpose of privilege verification.

$$\mathbf{VR} = \langle\{E_p = Y^{s_p}\}_{p\in\{1,\cdots,r-1\}}\rangle$$

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



(a) seen from the data consumers' perspective.

(b) seen from the server's perspective.

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

D. Decrypt($\mathbf{PK}$, $\mathbf{SK}_u$, $\mathbf{CT}$)

Every user within the system can download the ciphertext from the Cloud Server, but he is able to execute operations upon encrypted data only after he successfully decrypts it. Firstly, we define a recursive algorithm $DecryptNode(\mathbf{CT}, \mathbf{SK}_u, x)$, 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 $i \in \mathbb{A}^u$,

$$
\begin{aligned}
&DecryptNode(\mathbf{CT}, \mathbf{SK}_u, x) = \tfrac{e(D_i, C_x)}{e(D_i', C_x')} \\
&= \tfrac{e(g^{\sum v_k} \cdot H(att(i))^{r_i}, g^{q_x(0)})}{e(g^{r_i}, H(att(i))^{q_x(0)})} = e(g,g)^{(\sum v_k)\cdot q_x(0)}
\end{aligned}
$$

If not, we define $DecryptNode(\mathbf{CT}, \mathbf{SK}_u, x) := \perp$.

If x is not a leaf node, the algorithm proceeds as follows: For all nodes z that are children of x, it calls $DecryptNode(\mathbf{CT}, \mathbf{SK}_u, z)$ and stores the output as F_z. Let S_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \varnothing$. If no such set exists then the node was not satisfied and the algorithm returns $\perp$. Otherwise, compute

$$F_x := \prod_{z \in S_z} F_z^{\triangle_{d,s'_x}(0)}, \text{where} \begin{cases} d = index(z) \\ S'_x = index(z) : z \in S_x \end{cases}$$
$$= \prod_{z \in S_z} \left(e(g,g)^{(\sum v_k)\cdot q_z(0)}\right)^{\triangle_{d,S'_x}(0)}$$
$$= \prod_{z \in S_z} \left(e(g,g)^{(\sum v_k)\cdot q_{parent(z)}(d)}\right)^{\triangle_{d,S'_x}(0)}$$
$$= \prod_{z \in S_z} \left(e(g,g)^{(\sum v_k)\cdot q_x(d)}\right)^{\triangle_{d,S'_x}(0)}$$
$$= e(g,g)^{(\sum v_k)\cdot q_x(0)} \text{(using polynomial interpolation)}$$

The interpolation above recovers the parent node's value by calculating coefficients of the polynomial and evaluating the $p(0)$. Note that for a polynomial of node x with degree d_x, we need $d_x + 1$ known secret shares to calculate the coefficients, which is the threshold value k_x of the node. We direct the readers to [21] for complete calculation.

After a user downloads the ciphertext, he recursively call this algorithm, starting from the root node R_p of the tree T_p. If the tree is satisfied, which means he is authorized to the privilege p, then

$$DecryptNode(\mathbf{CT}, \mathbf{SK}_u, R_p) = e(g,g)^{s_p(\sum v_k)} = Y^{s_p}$$

Finally, if the user is trying to read the file, the symmetric encryption key K_e can be recovered by $K_e = \frac{E_0}{Y^{s_0}}$, and the data file can be decrypted by using it. Otherwise, if a user wants to execute a specific operation on a data file, he should be verified as an authorized user for the operation first. If the operation requires the j^{th} privilege, the user recursively calls $Decrypt(\mathbf{CT}, \mathbf{SK}_u, x)$ starting from the root node R_j of the tree T_j to get Y^{s_j}. The user sends it to the Cloud Server as well as the operation request. The Cloud Server checks whether $Y^{s_j} = E_j$, and proceeds if they do equal each other.

In fact, Y^{s_j} should be encrypted to avoid replay attack. This can be simply implemented by introducing any public key encryption protocol.

E. *ReEncrypt*($\mathbf{PK}$, $\mathbf{CT}$, $\mathbf{SK}_u$, $\{T'_p\}_{p \in \{0,\cdots,r-1\}}$)

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 avoid 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 not 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, $ReEncrypt(\mathbf{PK}, \mathbf{CT}, \mathbf{SK}_u, \{T'_p\})$ is executed to obtain new $\mathbf{CT}'$ and $\mathbf{VT}'$.

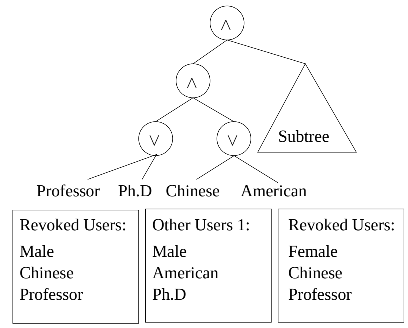


Fig. 4. An example of privilege tree after the re-encryption

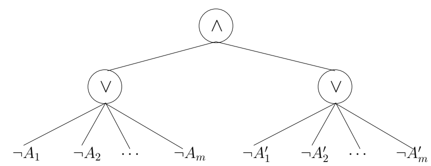


Fig. 5. An example of feasible subtree to prohibit two revoked users' access

Figure.4 shows a new tree T'_p after a re-encryption. Assuming 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 $\vee$ Ph.D Student).

V. Security Analysis

We first provide an informal security analysis for *Anony-Control*, then we give our formal proof of the construction.

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. Compromise Tolerance

In the proposed scheme, an authority $\mathcal{A}_k$ generates a set of random secret parameters $\{s_{kj}\}$ and shares it with another authority $\mathcal{A}_j$, 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 $g^{\sum v_k}$, and he fails to construct a valid secret key. Hence, the scheme achieves compromise tolerance to up to $(N-2)$ authorities compromise.

However, the compromised authorities are able to issue valid attribute keys 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.

C. Data Confidentiality against Collusion Attack

In order to access a plaintext, attackers must recover $Y^{s_0} = e(g,g)^{s_0 \sum v_k}$. This can be recovered if and only if the attackers have enough attributes to satisfy the tree T_0

since the value is randomized by attribute authorities. 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, a ciphertext cannot be decrypted by paring several private keys unless one of the keys does satisfy the privilege tree.

Security Model for *AnonyControl* construction

Without loss of generality, we assume there is only one privilege tree T_0, which represents the privilege for reading, in the privilege set of $\mathbf{CT}$. This assumption does not affect the security proof since we can easily extend the model to have several trees in $\mathbf{CT}$. 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 $\mathbf{PK}$ to the adversary.

Phase 1 The adversary queries for as many private keys, which correspond to attribute sets $\mathbb{A}_1, \cdots, \mathbb{A}_q$, 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 $\mathbf{CT}^*$ 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 $\mathcal{A}$ 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 advantage 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 $\mathcal{A}$ can attack our scheme in the security model above with advantage ϵ. We prove that the following DBDH game can be solved with advantage $\frac{\epsilon}{2}$.

Let $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_T$ be a bilinear map, where $\mathbb{G}_0$ 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 $\mu = 0$; otherwise he sets $(g, A, B, C, Z) := (g, g^a, g^b, g^c, e(g, g)^z)$, where $a, b, c, z \in \mathbb{Z}_p$ are randomly picked. The challenger then gives the simulator $\langle g, A, B, C, Z\rangle = \langle g, g^a, g^b, g^c, Z\rangle$. The simulator sim then plays the role of a challenger in the following DBDH game.

Init The adversary $\mathcal{A}$ 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 $\mathcal{A}$.

Phase 1 $\mathcal{A}$ queries for as many private keys, which correspond to attribute sets $\mathbb{A}_1, \cdots, \mathbb{A}_q$, 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 $\mathcal{A}$'s requests. For all attributes $i \in \mathbb{A}^u$, he randomly picks $r_i \in \mathbb{Z}_p$, and computes $D_i := A \cdot H(att(i))^{r_i}, D_i' := g^{r_i}$. Then, sim returns the created private key to $\mathcal{A}$.

Challenge The adversary $\mathcal{A}$ 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 $\mathcal{A}$.

$$\mathbf{CT}^* = \langle T_0^*, E_0 = m_\gamma \cdot Z, \\ \{C_i = g^{q_i(0)}, C_i' = H(att(i))^{q_i(0)}\}_{i \in \mathbb{A}^{T_0^*}}\rangle$$

If $\mu = 0$, $Z = e(g, g)^{abc}$. If we let $ab = \sum v_k$ and $c = s_0$ (this is possible because $v_k, k \in \{1, 2, \cdots, N\}$ and s_0 are all randomly chosen from $\mathbb{Z}_p$), we have $Z = e(g, g)^{abc} = (e(g, g)^{ab})^c = Y^{s_0}$. Therefore, $\mathbf{CT}^*$ is a valid ciphertext of the message m_γ. Otherwise, if $\mu = 1$, $Z = e(g, g)^z$. Then, we have $E_0 = m_\gamma \cdot e(g, g)^z$. Since $z \in \mathbb{Z}_p$ is a random element, E_0 is a random element in $\mathbb{G}_T$ from $\mathcal{A}$'s perspective, therefore $\mathbf{CT}^*$ contains no information about m_γ.

Phase 2 Repeat Phase 1 adaptively.

Guess $\mathcal{A}$ 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 $\mu = 1$, the adversary $\mathcal{A}$ learns no information about γ, so we have $\Pr[\gamma \neq \gamma' | \mu = 1] = \frac{1}{2}$. Since the challenger guesses $\mu' = 1$ when $\gamma = \gamma'$, we have $\Pr[\mu' = \mu | \mu = 1] = \frac{1}{2}$. If $\mu = 0$, the adversary $\mathcal{A}$ gets a valid ciphertext of m_γ. $\mathcal{A}$'s advantage in this situation is ϵ by definition, so we have $\Pr[\gamma = \gamma' | \mu = 0] = \frac{1}{2} + \epsilon$. Since the challenger guesses $\mu' = 0$ when $\gamma = \gamma'$, we have $\Pr[\mu' = \mu | \mu = 0] = \frac{1}{2} + \epsilon$. The overall advantage in this DBDH game is $\frac{1}{2}\Pr[\mu' = \mu | \mu = 0] + \frac{1}{2}\Pr[\mu' = \mu | \mu = 1] - \frac{1}{2}$ $= \frac{1}{2} \cdot (\frac{1}{2} + \epsilon) + \frac{1}{2} \cdot \frac{1}{2} - \frac{1}{2} = \frac{\epsilon}{2}$.

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

Since our scheme relies on the assumption that no probabilistic polynomial algorithm can solve the DBDH problem 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 calculate the total and average (per authority) computation complexity for each algorithm. For convenience, we denote N to be the number of attribute authorities, I to be the size of the entire attribute set and X to be the number of nodes in a tree T_p.

A. Setup

When the system is setup, $\prod Y_k$ 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 $O(C^2 \cdot \frac{N}{C}) = O(C \cdot N)$, but C is a constant number, so $O(C \cdot N) = O(N)$. 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, $g^{\sum v_j}$ is computed by N authorities, and $D_i = H(att(i))^{r_i} \cdot g^{\sum v_j}$ is computed for I times by one attribute authority. Therefore, the total complexity of Attribute Key Generation is $O(N^2 + I \cdot N)$. In the Aggregation of Two Keys, a user aggregates the I components, thus the computation complexity of this operation is $O(I)$. So, the total complexity of this process is $O(N^2 + I \cdot N)$. 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 $O(X \cdot K)$.

D. Decryption

$DecryptNode$ 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 $O(X)$.

E. User Revocation

This operation has the same complexity as the addition of Decryption and Encryption, thus its complexity is $O(X \cdot K)$.

TABLE III
Complexity Comparison (per authority)

Process	Yu *et al.* [11]	Chase *et al.* [6]	Ours
Setup	$O(I)$	$O(1)$	$O(1)$
Key Generation	$O(X)$	$O(N+I)$	$O(N+I)$
Encryption	$O(I)$	$O(I)$	$O(X \cdot K)$
Decryption	$O(\max(X,I))$	$O(N \cdot I)$	$O(X)$
User Revocation	$O(I)$		$O(X \cdot K)$

VII. Implementation

In this section, we give the experimental result of our scheme, which is conducted on the prototype of our system. To the best of our knowledge, this is the first implementation of a multi-authority attribute based encryption scheme. Our toolkit provides five command line tools.

anonyabe-setup : Using this command, attribute authorities co-generate a public key and N master keys.

anonyabe-keygen : Given a public key and a master key, generates a part of private key for the attribute set it is responsible for.

anonyabe-enc : Given a public key, encrypts a file under r privilege trees.

anonyabe-dec : Given a private key, decrypts a file if possible.

anonyabe-rec : Given a public key and a private key, re-encrypts a file under other privilege trees.

This toolkit is based on the CP-ABE toolkit [22] which is in turn based on Pairing-Based Cryptography library [23], and the whole experiment system is implemented on a linux system with Intel i7 2^{nd} Gen @ 2.7GHz and 2GB RAM.

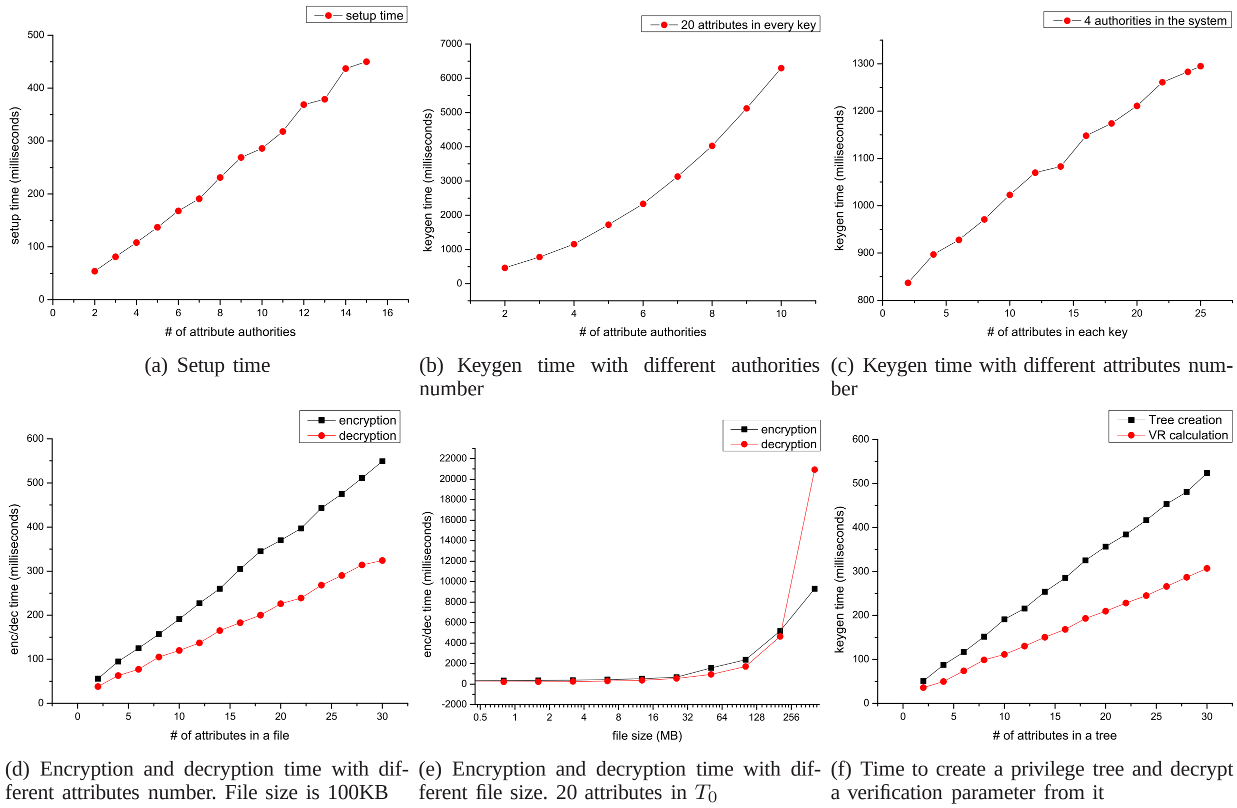
Figure.6 shows the computation overhead incurred in Setup, Key Generation, Encryption and Decryption, which are the core algorithms in our system under various conditions.

Figure.6(a) shows the total setup time (system-wide) with different number of attribute authorities. Figure.6(b) shows the total key generation time (system-wide) with different number of authorities, and the number of attributes is fixed to 20. Figure.6(c) shows the key generation time with different number of attributes in each key, and the number of authorities is fixed to 4. Figure6(d) shows the encryption and decryption time with different number of attributes in T_0, and we set only one privilege for file access to measure the most frequent operation, file access. Figure.6(e) shows the encryption and decryption time with different file size, where the number of attributes in T_0 is fixed to 20. Figure.6(f) shows the time to create a privilege tree for privilege j and calculate a verification parameter Y^{s_j} from it. Obviously, the total time needed to create one $\mathbf{VR}$ set is approximately equal to $r \cdot t$, where r is the number of total privileges and t is the time for creating one tree.

The Re-encryption is omitted because it is barely a composition of Decryption and Encryption. Interestingly, in a series of the experiment, the run time of encryption and decryption was independent of the tree structure. That is, no matter how complicated the tree is, the computation complexity of encryption and decryption depends only on the number of nodes in the tree, which coincides with the performance analysis table in the previous section. Results of other algorithms are just as we expected. The graphs generally followed the growth rate showed in the performance analysis table above.

VIII. Conclusion

This paper proposed an anonymous attribute-based privilege control scheme *AnonyControl* to address the user privacy problem in cloud computing. Using multiple authorities in



(a) Setup time

(b) Keygen time with different authorities number

(c) Keygen time with different attributes number

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

(e) Encryption and decryption time with different file size. 20 attributes in T_0

(f) Time to create a privilege tree and decrypt a verification parameter from it

Fig. 6. Experiment result on our implemented prototype system

the cloud computing system, the proposed scheme achieves not only anonymous cloud data control, but also fine-grained privilege control for cloud computing. More importantly, the proposed scheme can tolerate up to $(N-2)$ authority compromise, which is highly preferable in Internet-based cloud computing. 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 conduct detailed security and performance analysis which shows that *AnonyControl* is both secure and efficient for cloud computing.

References

[1] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in cryptology*. Springer, 1985, pp. 47–53.

[2] A. Sahai and B. Waters, "Fuzzy identity-based encryption," *Advances in Cryptology–EUROCRYPT 2005*, pp. 557–557, 2005.

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

[4] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*. Ieee, 2007, pp. 321–334.

[5] M. Chase, "Multi-authority attribute based encryption," *Theory of Cryptography*, pp. 515–534, 2007.

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

[7] J. Horwitz and B. Lynn, "Toward hierarchical identity-based encryption," in *Advances in CryptologyEUROCRYPT 2002*. Springer, 2002, pp. 466–481.

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

[9] T. Nishide, K. Yoneyama, and K. Ohta, "Attribute-based encryption with partially hidden encryptor-specified access structures," in *Applied Cryptography and Network Security*. Springer, 2008, pp. 111–129.

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

[11] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*. Ieee, 2010, pp. 1–9.

[12] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 735–737.

[13] J. Liu, Z. Wan, and M. Gu, "Hierarchical attribute-set based encryption for scalable, flexible and fine-grained access control in cloud computing," *Information Security Practice and Experience*, pp. 98–107, 2011.

[14] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM, 2010, pp. 261–270.

[15] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," *Advances in CryptologyEUROCRYPT'98*, pp. 127–144, 1998.

[16] A. Kapadia, P. Tsang, and S. Smith, "Attribute-based publishing with hidden credentials and hidden policies," *NDSS07*, pp. 179–192, 2007.

[17] S. Yu, K. Ren, and W. Lou, "Attribute-based content distribution with hidden policy," in *Secure Network Protocols, 2008. NPSec 2008. 4th Workshop on*. IEEE, 2008, pp. 39–44.

[18] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[19] C. Wang, K. Ren, and J. Wang, "Secure and practical outsourcing of linear programming in cloud computing," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 820–828.

[20] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*. IEEE, 2010, pp. 253–262.

[21] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[22] "Cp-abe toolkit," in *http://acsc.csl.sri.com/cpabe/*.

[23] "Pairing-based cryptography library," in *http://crypto.stanford.edu/pbc/*.