A RBAC Model Based on Identity-Based Cryptosystem in Cloud Storage

Jian Xu(✉), Yanbo Yu, Qingyu Meng, Qiyu Wu, and Fucai Zhou

Software College, Northeastern University, Shenyang 110169, China
xuj@mail.neu.edu.cn

Abstract. Aiming at the shortcomings of most of existing ciphertext access control scheme in cloud storage does not support dynamic update of access control strategy, has large computational overhead ,combine identity-based cryptosystem and role based access control model (using RBAC1 model of the RBAC96 model family), build RBAC model based on identity-based cryptosystem in cloud storage. This paper presents a formal definition of the scheme, a detailed description of four tuple used to represent access control strategy, the hybrid encryption strategy and Re-encrypt when writing strategy in order to improve the efficiency of the system, detailed steps of system initialization, add and delete users, add and delete permissions, add and delete roles, add and delete role inheritance, assign and remove user, assign and remove permission, read and write file algorithm.

Keywords: Access control · RBAC · Identity-based cryptosystem · Cloud storage

1 Introduction

With the rapid development of computer technology and Internet applications, data is growing at an exponential rate. Faced with such massive data, cloud storage which developed from the concept of cloud computing has become the most common and popular third-party storage with its advantages of low cost, huge capacity, resource sharing, easy management and good scalability. Users and enterprises can purchase storage services flexibly according to their own needs, which not only save expensive software and hardware infrastructure investment, but also ensure that storage resources are fully utilized. Cloud storage service providers also provide professional data backup, disaster recovery and other functions to effectively ensure the continuity of services [1, 2].

Although cloud storage has many advantages, however, its promotion is relatively slow. The main reason is that once the data is uploaded to the cloud, users lose control of it, and they do not know what the cloud storage provider will do with the data. Cloud storage providers may snoop on the content of the data and even provide the user's data directly to third parties, especially in an untrusted cloud environment. Therefore, ensuring the confidentiality of user data, avoiding it being illegally accessed, achieving secure and efficient access control is the key to solving data security problems in cloud storage [9]. However, as the cloud environment has the characteristics of dynamic

J. Li et al. (Eds.): SPNCE 2019, LNICST 284, pp. 362–377, 2019.
https://doi.org/10.1007/978-3-030-21373-2_28

change, multi-tenancy and virtualization, the traditional access control model cannot meet the requirements of the new cloud architecture. So how to expand and optimize the traditional access control model, especially combine advanced encryption technology with traditional access control models, to build an access control scheme for cloud storage environment has become a hot topic in academic research. In the case that the cloud storage provider is not trusted, the research of ensuring the confidentiality of data and implement access control of the ciphertext data is therefore a top priority.

1.1 Related Works

Many researchers have started research on access control technology under the cloud computing environment and have obtained many research results.

Jung et al. [3] proposed an adaptive resource access control scheme based on the RBAC model. This scheme can dynamically adjust the security level of resources and solve the problem of dynamic changes of environment variables in cloud computing. Wang et al. [4] introduced the concept of task into the RBAC model and proposed a task- and role-based access control model (T-RBAC) in the cloud computing environment. In T-RBAC, workflow is first broken down into a series of interdependent tasks, which are then assigned to the role, and the user gets the role by executing the task node. The Danwei et al. [5] adopts a negotiation policy when designing the access control model, and proposes a UCON-based cloud service access control scheme. Based on the UCON model, Krautsevich et al. [6] introduced a risk assessment mechanism, and purposed an access control scheme for highly dynamic system, which improves the flexibility and security of the UCON model.

Attribute-Based Encryption (ABE) is the most commonly used advanced encryption algorithm in cloud access control. It extends the concept of identity as a set of attributes. In 2005, Sahai et al. [7] first proposed a fuzzy identity-based encryption scheme, in which the concept of attributes was introduced. Subsequently, based on this, Goyal et al. [8] proposed an attribute-based encryption scheme. And then derived two attribute-based encryption algorithms closely related to the access control policy, KP-ABE (Key Policy Attribute-Based Encryption) [9] and CP-ABE (Ciphertext Policy Attribute-Based Encryption) [10], in which CP-ABE is more suitable for cloud environments. Sun et al. [11] proposed a data security access control scheme for cloud storage based on CP-ABE. This scheme adopts a distribution method for distributed key, and access control is implemented by designing keys. However, the scheme is suitable for the case where the access permission type is small, once the type is increased, key management may become very complicated. Jung et al. [12] designed an anonymous access control scheme, which perform anonymous access control on cloud data to protect user's privacy; Ruj et al. [13] proposes a cloud security access control framework that can implement user authentication and privacy protection of data.

Although the above attribute-based access control schemes can ensure the confidentiality of user data and achieve fine-grained access control of data, but they do not support dynamic update of access control policy. It obviously does not meet the requirements of the dynamic environment in the cloud. For the problems mentioned above, researchers have proposed some schemes. Yu et al. [14] proposed a secure and

scalable cloud storage access control scheme based on CP-ABE, which supports attribute revoking and employs a proxy re-encryption policy to save computational overhead; Hur et al. [15] designed an access control scheme that supports user attribute revoking, using double-layer encryption mechanism to improve efficiency; Chen et al. [16] proposed a hybrid access control scheme that supports hierarchical management of multiple authorization mechanisms, which reduced management complexity. Although these schemes can support the revoke operation, the computational overhead cannot be ignored.

As mentioned above, there are many shortcomings of current access control schemes in the cloud environment. Although simply optimize the traditional access control model and apply it to the cloud environment can implement basic access control functions, the confidentiality of data cannot be guaranteed. The attribute-based ciphertext access control scheme can protect data when the cloud storage provider is not trusted, but most schemes do not support the dynamic update of access control policy, or the computational overhead is huge although the policy update is supported.

1.2 Contributions

To solve the problems mentioned above, this paper proposes an RBAC scheme based on identity cryptosystem in cloud storage. This paper describes the application scenarios and entity composition of the scheme, and gives a formal definition of the scheme. The four tuples used to represent the access control policy in the scheme are described in detail, as well as the hybrid encryption policy and the write-time re-encryption policy designed to improve system efficiency. Detailed steps of system initialization, user addition and deletion, permission addition and deletion, role addition and deletion, role's inheritance relationship addition and deletion, user assignment and revocation, permission assignment and revocation, file reading and writing are given.

2 Constructions

2.1 Design Idea

Our scheme is mainly composed of three-party entities: access control administrators, cloud storage providers (reference monitors are deployed in the cloud as part of the cloud storage provider, not separately listed as one entity) and users. The roles and functions of the three entities are as follows:

(1) Access control administrator: It is the administrator of the access control system, responsible for developing and updating access control policy. It determines who has the permission to access the resource. In this scheme, it has two important functions. One is to act as a key generation center in the identity-based cryptosystem; it holds the system master key, creates and distributes keys for users in the system. Second is to develop and update the access control policy in the system. The specific operations include the addition and deletion of roles, the addition and deletion of role's inheritance relationships, the assignment and revocation of users, and the assignment and revocation of permissions.

(2) Cloud storage provider: As a provider of storage services, it manages the storage needs of users. It not only stores the data that users deposit in the cloud, but also stores access control policy that provide protection for those data. This paper assumes that the cloud storage provider is untrustworthy and cannot let it view the contents of the file stored on it, but at the same time believes that it can guarantee the availability of the file and that only authorized users can change the content of the file.
Reference monitor: Most access control models have one thing in common, that is, relying on a trusted reference monitor to check whether an access request conforms to an access control policy before accessing the protected resource. In this scheme, the reference monitor is deployed in the cloud and is responsible for coordinating authorized access to resources. For example, when write permission is executed, it is responsible for verifying that if the user's signature is valid and checking if the user has write permission.
In a nutshell, cloud storage providers ensure file system consistency by blocking unauthorized updates, while it cannot read files or change files and access control policy.
(3) User: It is a user of the cloud storage service and is managed by the access control administrator. It needs to register with the administrator and obtain its own key before using the system. It can upload its own data to the cloud storage, or download the data in the cloud for read and write operations (The premise is that there is a corresponding access permission, otherwise the data will not be decrypted, or the reference monitor will determine that there is no corresponding permission, and the operation cannot be performed).

2.2 Formal Definitions

Definition 1: The RBAC scheme based on identity cryptosystem (RBAC-IBC) in cloud storage can be represented by a tuple composed of eight PPT algorithms, that is, RBAC-IBC = $(Setup, User, Permission, Role, Inh, UR, PA, R\&W)$. The details are described as follows:

(1) $Setup(I^n)$: System initialization algorithm. The input is security parameter n, which generates a common parameter of identity-based encryption algorithm and a master key of identity-based signature algorithm, and generates an identity-based decryption key and signature key for the administrator.
(2) $User(addU(u), delU(u))$: User addition and deletion algorithm. It contains two sub-algorithms: user addition algorithm and user deletion algorithm. The input of user addition algorithm is the username u, which generates an identity-based decryption key and signature key for the user; the input of user deletion algorithm is also the username u, which revokes the user from the system, and then the user will not be able to access any files in the cloud.
(3) $Permission(addP(fn, f), delP(fn))$: Permission addition and deletion algorithm. It contains two sub-algorithms: permission addition algorithm and permission deletion algorithm. The input of permission addition algorithm is the file name fn

and the file content f, which encrypts the file and uploads it to the cloud; the input of permission deletion algorithm is the file name *fn*, which removes the file from the system.

(4) $Role(addR(r), delR(r))$: Role addition and deletion algorithm. It contains two sub-algorithms: role addition algorithm and role deletion algorithm. The input of role addition algorithm is the role name r, which adds a role to the access control system and generates an identity-based decryption key and signature key for the role; the input of role deletion algorithm is the role name r, which revokes the role from the system.

(5) $Inh(addInh(r_c, r_p), delInh(r_c, r_p))$: Role's inheritance relationship addition and deletion algorithm. It contains two sub-algorithms: role's inheritance relationship addition algorithm and role's inheritance relationship deletion algorithm. The input of role's inheritance relationship addition algorithm is the child role r_c and the parent role r_p, which adds a role's inheritance relationship to the system, so that the role r_c inherits the role r_p; the input of role's inheritance relationship deletion algorithm is the child role r_c and the parent role r_p, which revokes the inheritance relationship between them from the system.

(6) $UR(assignU(r, u), revokeU(r, u))$: User assignment and revocation algorithm. It contains two sub-algorithms: user assignment algorithm and user revocation algorithm. The input of user assignment algorithm is the role r and the user u, which assigns user u to role r; the input of user revocation algorithm is the role r and the user u, which revokes user u from the user list of role r.

(7) $PA(assignP(r, \langle fn, op\rangle), revokeP(r, \langle fn, op\rangle))$: Permission assignment and revocation algorithm. It contains two sub-algorithms: permission assignment algorithm and permission revocation algorithm. The input of permission assignment algorithm is the role r, the file name *fn* and permission name *op*, which assigns the operation permission (*op*) of the file *fn* for the role r; the input of permission revocation algorithm is the role r, the file name *fn* and permission name *op*, which revokes the operation permission (*op*) of the file *fn* from the role r.

(8) $R\&W(read(fn), write(fn, f))$: File reading and writing algorithm. It contains two sub-algorithms: file reading algorithm and file writing (update) algorithm. The input of file reading algorithm is the file name *fn*, and the user reads file; the input of file writing algorithm is the file name *fn* and the updated content of the file f, which updates file content stored in the cloud.

According to the access control scheme evaluation method, the relevant properties used to evaluate RBAC-IBC are defined as follows:

Definition 2: If the implementation of the RBAC scheme based on the identity cryptosystem $\langle \sigma, \alpha, \pi \rangle$ has the following properties:

Property 1: Command mapping protects state mapping and protects security.
Property 2: State mapping protects query mapping.
Property 3: Query mapping is access control protected.
Then the scheme is correct, access control protected, and secure.

2.3 Detailed Description

The scheme is divided into eight parts by function: System initialization, user addition and deletion, permission addition and deletion, role addition and deletion, role's inheritance relationship addition and deletion, user assignment and revocation, permission assignment and revocation, file reading and writing. Each detailed step is given below.

For convenience, first explain the symbols used in this section. The definition of each symbol is described in Table 1.

Table 1. Symbol description

Symbol	Description
u	User name
r	Role name
f	File (Here is the file content itself)
fn	File name
v	Version number
k	Symmetric key
EK	Identity-based decryption key
SK	Identity-based signature key
$USERS$	File that stores the username
$ROLES$	File that stores the role name and role's key version number
$FILES$	File that stores the file name and file's key version number
SU	Access control administrator
$R.M$	Reference monitor deployed in the cloud
–	Wildcard

2.3.1 System Initialization

$Setup(I^n)$: System initialization algorithm. The administrator performs system initialization operation. The main steps are as follows:

Step 1: Perform the initialization algorithm of identity-based encryption and identity-based signature scheme. Generate their own public parameters and master keys, expose public parameters, and secretly save the master keys.
Step 2: Create three empty files——$USERS$, $ROLES$ and $FILES$, upload $ROLES$ and $FILES$ to the cloud.
Step 3: Generate an identity-based decryption key EK_{SU} and signature key SK_{SU} for himself:

$$KeyGen^{IBE}(SU) \rightarrow EK_{SU}, KeyGen^{IBS}(SU) \rightarrow SK_{SU}.$$

Note that in order to save space, the description of the key generation process is simplified here, and parameters such as the master keys are not listed, and the subsequent algorithm description is also the same.

2.3.2 User Addition and Deletion

$addU(u)$: User addition algorithm. When a new user joins the system, he needs to register with the administrator and the administrator performs the user addition operation. The main steps are as follows:

Step 1: Add the username u to the file *USERS*.
Step 2: Generate the identity-based decryption key EK_u and signature key SK_u for the user:

$$KeyGen^{IBE}(u) \rightarrow EK_u, KeyGen^{IBS}(u) \rightarrow SK_u.$$

Step 3: Send EK_u and SK_u to the user via the trusted channel

$delU(u)$: User deletion algorithm. To delete a user from the system, the administrator needs to do the following operations:

For each role r that contains user u, perform the operation $*revokeU(r,u)$.
The symbol $*$ here means that the specific steps of this operation will be given later, and the following appears $*$ is synonymous with this.

2.3.3 Permission Addition and Deletion

$addP_u(fn,f)$: Permission addition algorithm. Actually, the operation of adding permission is that the user uploads file. The permission to read and write files is first assigned to the administrator, who then assigns permission to the role. The main steps are as follows:

Step 1: User generates the symmetric key required to encrypt the file: $KeyGen^{Sym} \rightarrow k$.
Step 2: Build tuple F and tuple PA (the initial file key version number is set to 1), the forms are as follows: $\langle F, fn, 1, Enc_k^{Sym}(f), u, Sign_u^{IBS}\rangle$, $\langle PA, SU, (fn, RW), 1, Enc_{SU}^{IBE}(k), u, Sign_u^{IBS}\rangle$, and send them to the cloud.
Step 3: After receiving the two tuples, the reference monitor *R.M* deployed in the cloud performs the following operations:

(1) Check that the tuple format is correct. If the format is correct, proceed to the next step, otherwise send an error report.
(2) Verify that the user's identity-based signature is legal. If the signature is legal, that is:
$Verify_u^{IBS}(\langle F, fn, 1, Enc_k^{Sym}(f), u\rangle, Sign_u^{IBS}) = 1$
$Verify_u^{IBS}(\langle PA, SU, (fn, RW), 1, Enc_{SU}^{IBE}(k), u\rangle, Sign_u^{IBS}) = 1$
proceed to the next step, otherwise send an error report.
(3) Add the file name and file's key version number $(fn, 1)$ to the file *FILES*, store the two tuples to the appropriate location in the cloud.

delP(fn): Permission deletion algorithm. The deletion of permission is actually to delete all tuples related to a file stored in the cloud. The administrator notifies *R.M* to perform it. The main steps are as follows:

Step 1: *R.M* deletes (fn, v_{fn}) from file *FILES*.
Step 2: Delete all $\langle F, fn, -, -, -, - \rangle$ tuples and $\langle PA, -, (fn, -), -, -, -, - \rangle$ tuples.

2.3.4 Role Addition and Deletion

addR(r): Role addition algorithm. The administrator adds a role to the system. The main steps are as follows:

Step 1: Add role name and the initial version number $(r, 1)$ of role key to the file *ROLES*.
Step 2: Generate the identity-based decryption key $EK_{(r,1)}$ and signature key $SK_{(r,1)}$ for the role.
Step 3: Build tuple $\langle UR, SU, (r, 1), Enc_{SU}^{IBE}(EK_{(r,1)}, SK_{(r,1)}), Sign_{SU}^{IBS} \rangle$ and send it to *R.M*.
Step 4: *R.M* stores the received tuple to the corresponding location in the cloud.

Note that the initial version number of the role key is set to 1 here. And because the administrator needs to assign users to roles in the future, the administrator first becomes a member of the role so that the key of the role can be accessed later.

delR(r): Role deletion algorithm. The administrator deletes the role in the system. The main steps are as follows:

Step 1: Revoke (r, v_r) from file *ROLES*, delete all tuples $\langle UR, -, (r, v_r), -, - \rangle$.
Step 2: If role r is a parent role, delete all tuples $\langle RH, -, (r, v_r), -, - \rangle$; if role r is a child role, delete all tuples $\langle RH, (r, v_r), -, -, - \rangle$, and perform operation $*delInh(r, -)$ on all of its parent roles.
Step 3: For each permission $\langle fn, op \rangle$ owned by role r, perform operation $*revokeP(r, \langle fn, RW \rangle)$.

2.3.5 Role's Inheritance Relationship Addition and Deletion

addInh(r_c, r_p): Role's inheritance relationship addition algorithm. The administrator makes role r_c inherit role r_p, let r_c have all permissions of r_p. The main steps are as follows:

Step 1: Download tuple $\langle UR, SU, (r_p, v_{r_p}), Enc_{SU}^{IBE}(EK_{(r_p,v_{r_p})}, SK_{(r_p,v_{r_p})}), Sign_{SU}^{IBS} \rangle$ from the cloud and verify the signature.
If $Verify_{SU}^{IBS}(\langle UR, SU, (r_p, v_{r_p}), Enc_{SU}^{IBE}(EK_{(r_p,v_{r_p})}, SK_{(r_p,v_{r_p})}) \rangle, Sign_{SU}^{IBS}) = 1$, proceed to the next step, otherwise send an error report.
Step 2: The administrator decrypts the decryption key and signature key of the parent role from the *UR* tuple using his own decryption key EK_{SU}:

$$(EK_{(r_p,v_{r_p})}, SK_{(r_p,v_{r_p})}) = Dec^{IBE}_{EK_{SU}}(Enc^{IBE}_{SU}(EK_{(r_p,v_{r_p})}, SK_{(r_p,v_{r_p})})).$$

Step 3: Build tuple $\langle RH, (r_c, v_{r_c}), (r_p, v_{r_p}), Enc^{IBE}_{(r_c,v_{r_c})}(EK_{(r_p,v_{r_p})}, SK_{(r_p,v_{r_p})}), Sign^{IBS}_{SU}\rangle$ and send it to *R.M*.

Step 4: *R.M* stores the received tuple to the corresponding location in the cloud.

delInh(r_c, r_p): Role's inheritance relationship deletion algorithm. The administrator deletes the inheritance relationship between role r_c and role r_p. For the parent role, it is equivalent to remove a user from it. The main steps are as follows:

Step 1: Delete tuple $\langle RH, (r_c, v_{r_c}), (r_p, v_{r_p}), Enc^{IBE}_{(r_c,v_{rc})}(EK_{(r_p,v_{r_p})}, SK_{(r_p,v_{r_p})}), Sign^{IBS}_{SU}\rangle$.

Step 2: Generate a new decryption key and signature key for the parent role: $KeyGen^{IBE}((r_p, v_{r_p}+1)) \rightarrow EK_{(r_p,v_{r_p}+1)}$, $KeyGen^{IBS}((r_p, v_{r_p}+1)) \rightarrow SK_{(r_p,v_{r_p}+1)}$, update the file *ROLES* to increase the role's key version number by 1.

Step 3: Generate a new *RH* tuple for the other child roles ($r'_c \neq r_c$) of the parent role: That is, $\langle RH, (r'_c, v_{r'_c}), (r_p, v_{r_p}+1), Enc^{IBE}_{(r'_c,v_{r'_c})}(EK_{(r_p,v_{r_p}+1)}, SK_{(r_p,v_{r_p}+1)}), Sign^{IBS}_{SU}\rangle$, and upload it to *R.M* and replace the old tuples.

Step 4: Generate a new *UR* tuple for all user members of the parent role. That is, build a new tuple $\langle UR, -, (r_p, v_{r_p}+1), Enc^{IBE}_{-}(EK_{(r_p,v_{r_p}+1)}, SK_{(r_p,v_{r_p}+1)}), Sign^{IBS}_{SU}\rangle$ and upload it to *R.M*.

Step 5: Generate a new *PA* tuple for all files that the parent role can access, the specific steps are as follows:

(1) The administrator first downloads tuple $\langle UR, SU, (r_p, v_{r_p}), Enc^{IBE}_{SU}(EK_{(r_p,v_{r_p})}, SK_{(r_p,v_{r_p})}), Sign^{IBS}_{SU}\rangle$ from the cloud and verifies the signature, proceed to next step if the verification is passed.

(2) The administrator uses his own decryption key EK_{SU} to decrypt the role's decryption key and signature key from the *UR* tuple:

$$(EK_{(r_p,v_{r_p})}, SK_{(r_p,v_{r_p})}) = Dec^{IBE}_{EK_{SU}}(Enc^{IBE}_{SU}(EK_{(r_p,v_{r_p})}, SK_{(r_p,v_{r_p})})).$$

(3) For each tuple $\langle PA, (r_p, v_{r_p}), (fn, op), v_{fn}, Enc^{IBE}_{(r_p,v_{r_p})}(k), SU, Sign^{IBS}_{SU}\rangle$, first use role's decryption key to decrypt the file's symmetric key: $k = Dec^{IBE}_{EK_{(r_p,v_{r_p})}}(Enc^{IBE}_{(r_p,v_{r_p})}(k))$. Then build a new tuple $\langle PA, (r_p, v_{r_p}+1), (fn, op), v_{fn}, Enc^{IBE}_{(r_p,v_{r_p}+1)}(k), SU, Sign^{IBS}_{SU}\rangle$ and upload it to *R.M*.

(4) Delete all $\langle PA, (r_p, v_{r_p}), -, -, -, -, -\rangle$ tuples and $\langle UR, -, (r_p, v_{r_p}), -, -\rangle$ tuples.

Step 6: Update the symmetric key of all files that parent role can access, the specific steps are as follows:

(1) Generate a new symmetric key for each file that parent role can access: $KeyGen^{Sym} \rightarrow k'$.
(2) Generate a new *PA* tuple for all roles that have access to the above files: build a new tuple $\langle PA, -, (fn, op), v_{fn}+1, Enc_{-}^{IBE}(k'), SU, Sign_{SU}^{IBS}\rangle$ and upload it to *R.M*.
(3) Update file *FILES*, make the symmetric key version number of the file add 1.

Note that the write-time re-encryption policy is used in the role's inheritance relationship deletion algorithm. The specific embodiment is: After step 6 is executed, the cloud actually stores two versions of the *PA* tuple of files that the parent role can access. The only difference between the two versions is that the file key version number and the encrypted file's symmetric key is different. When the file is read, the old symmetric key is used for decryption; When the file is written, the new symmetric key is used for encryption. This is the specific implementation of write-time re-encryption policy.

2.3.6 User Assignment and Revocation

$assignU(r, u)$: User assignment algorithm. The administrator assigns users to the roles. The main steps are as follows:

Step 1: Download tuple $\langle UR, SU, (r, v_r), Enc_{SU}^{IBE}(EK_{(r,v_r)}, SK_{(r,v_r)}), Sign_{SU}^{IBS}\rangle$ from the cloud and verify the signature. If $Verify_{SU}^{IBS}(\langle UR, SU, (r, v_r), Enc_{SU}^{IBE}(EK_{(r,v_r)}, SK_{(r,v_r)})\rangle, Sign_{SU}^{IBS}) = 1$, proceed to the next step.
Step 2: The administrator uses his own decryption key EK_{SU} to decrypt the role's decryption key and signature key from the *UR* tuple: $(EK_{(r,v_r)}, SK_{(r,v_r)}) = Dec_{EK_{SU}}^{IBE}(Enc_{SU}^{IBE}(EK_{(r,v_r)}, SK_{(r,v_r)}))$.
Step 3: Build tuple $\langle UR, u, (r, v_r), Enc_{u}^{IBE}(EK_{(r,v_r)}, SK_{(r,v_r)}), Sign_{SU}^{IBS}\rangle$ and upload it to *R.M*.
Step 4: *R.M* stores the received tuple to the corresponding location in the cloud.

$revokeU(r, u)$: User revocation algorithm. The administrator revokes a user from the role. The main steps are as follows:

Step 1: Generate a new decryption key and signature key for role *r*: $KeyGen^{IBE}((r, v_r+1)) \rightarrow EK_{(r,v_r+1)}$, $KeyGen^{IBS}((r, v_r+1)) \rightarrow SK_{(r,v_r+1)}$. Update file *ROLES*, make the key version number of the role add 1.
Step 2: If role *r* has child roles, generate a new *RH* tuple for all its child roles. That is, build a new tuple $\langle RH, -, (r, v_r+1), Enc_{-}^{IBE}(EK_{(r,v_r+1)}, SK_{(r,v_r+1)}), Sign_{SU}^{IBS}\rangle$ and upload it to *R.M* and replace all old tuples.
Step 3: Generate a new *UR* tuple for other user members $(u' \neq u)$ of the role. That is, build a new tuple $\langle UR, u', (r, v_r+1), Enc_{u'}^{IBE}(EK_{(r,v_r+1)}, SK_{(r,v_r+1)}), Sign_{SU}^{IBS}\rangle$ and upload it to *R.M*.
Step 4: Generate a new *PA* tuple for files that all roles can access. The specific steps are as follows:

(1) First download tuple $\langle UR, SU, (r, v_r), Enc_{SU}^{IBE}(EK_{(r,v_r)}, SK_{(r,v_r)}), Sign_{SU}^{IBS}\rangle$ from the cloud and verify the signature. If verification passed, proceed to the next step.

(2) The administrator uses his own decryption key EK_{SU} to decrypt the role's decryption key and signature key from the UR tuple: $(EK_{(r,v_r)}, SK_{(r,v_r)}) = Dec^{IBE}_{EK_{SU}}(Enc^{IBE}_{SU}(EK_{(r,v_r)}, SK_{(r,v_r)}))$.
(3) For each tuple $\langle PA, (r, v_r), (fn, op), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$, first use role's decryption key to decrypt the file's symmetric key: $k = Dec^{IBE}_{EK_{(r,v_r)}}(Enc^{IBE}_{(r,v_r)}(k))$. Then build a new tuple $\langle PA, (r, v_r + 1), (fn, op), v_{fn}, Enc^{IBE}_{(r,v_r+1)}(k), SU, Sign^{IBS}_{SU}\rangle$ and upload it to $R.M$.
(4) Delete all $\langle PA, (r, v_r), -, -, -, -, -\rangle$ tuples and $\langle UR, -, (r, v_r), -, -\rangle$ tuples.

Step 5: Generate a new symmetric key for the files that all roles can access: $KeyGen^{Sym} \rightarrow k'$.
Step 6: Generate a new PA tuple for all roles that have access to the files in step 5. That is, for all $\langle PA, -, (fn, op), v_{fn}, Enc^{IBE}_{-}(k), SU, Sign^{IBS}_{SU}\rangle$, perform the following operations:

(1) Build a new tuple $\langle PA, -, (fn, op), v_{fn} + 1, Enc^{IBE}_{-}(k'), SU, Sign^{IBS}_{SU}\rangle$ and upload it to $R.M$.
(2) Update file $FILES$, make the symmetric key version number of file add 1.

The user revocation algorithm also uses the write-time re-encryption policy. After step 6 is executed, the cloud also stores two versions of the PA tuple of the files that the roles can access.

2.3.7 Permission Assignment and Revocation

$assignP(r, \langle fn, op\rangle)$: Permission assignment algorithm. The administrator assigns permissions to the role. The main steps are as follows:

Step 1: If the role already has read access to the file and needs to add write access, i.e. $op = RW$ and $\langle PA, (r, v_r), (fn, R), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$ already exists, perform the following operations:

(1) Download all versions of $\langle PA, (r, v_r), (fn, R), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$ and verify the signatures of tuples. If $Verify^{IBS}_{SU}(\langle PA, (r, v_r), (fn, R), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU\rangle, Sign^{IBS}_{SU}) = 1$, proceed to the next step.
(2) Build the new tuple $\langle PA, (r, v_r), (fn, RW), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$, upload it to $R.M$ and replace old tuples.

Step 2: If the role does not have any permissions on the file, first download the tuple $\langle PA, SU, (fn, RW), v_{fn}, Enc^{IBE}_{SU}(k), SU, Sign^{IBS}_{SU}\rangle$ from the cloud and verify the signature. If $Verify^{IBS}_{SU}(\langle PA, SU, (fn, RW), v_{fn}, Enc^{IBE}_{SU}(k), SU\rangle, Sign^{IBE}_{SU}) = 1$, proceed to next step.

(1) Decrypt the symmetric key of the file from PA tuple: $k = Dec^{IBE}_{EK_{SU}}(Enc^{IBE}_{SU}(k))$.

(2) Build the new tuple $\langle PA, (r, v_r), (fn, op), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$ and upload it to the cloud. After receiving the tuple, *R.M* stores it in the corresponding location in the cloud.

revokeP$(r, \langle fn, op\rangle)$: Permission revocation algorithm. The administrator revokes a permission from the role. The main steps are as follows:

Step 1: If only remove write permission and retain read permission, i.e. $op = W$, perform the following operations: Download all versions of tuple $\langle PA, (r, v_r), (fn, RW), -, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$ from the cloud and verify the signature. If $Verify^{IBS}_{SU}(\langle PA, (r, v_r), (fn, RW), -, Enc^{IBE}_{(r,v_r)}(k), SU\rangle, Sign^{IBS}_{SU}) = 1$, build the new tuple $\langle PA, (r, v_r), (fn, R), -, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$, upload it to *R.M* and replace the old tuple.

Step 2: If read and write permissions are revoked, i.e. $op = RW$, then:

(1) Delete all $\langle PA, (r, v_r), (fn, RW), -, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$ tuples.
(2) Generate a new symmetric key for the file: $KeyGen^{Sym} \rightarrow k'$.
(3) Generate new *PA* tuples for all other roles that can access the files. That is, build a new tuple $\langle PA, -, (fn, op), v_{fn} + 1, Enc^{IBE}_{-}(k'), SU, Sign^{IBS}_{SU}\rangle$ and upload it to *R.M*.
(4) Update file *FILES*, make the symmetric key version number of file add 1.

Note that in the step 2 of permission revocation algorithm, the write-time re-encryption policy is also used. In step 3, a new version of *PA* tuple containing the file's new symmetric key is generated for the role, which is stored in the cloud along with the *PA* tuple containing the file's old symmetric key.

2.3.8 File Reading and Writing

read$_u(fn)$: File reading algorithm. User reads a file, the main steps are as follows:

First, the user downloads the tuple $\langle F, fn, v_{fn}, Enc^{Sym}_{k}(f), (r, v_r), Sign^{IBS}_{(r,v_r)}\rangle$ from the cloud and verifies the signature. If

$Verify^{IBS}_{(r,v_r)}(\langle F, fn, v_{fn}, Enc^{Sym}_{k}(f), (r, v_r)\rangle, Sign^{IBS}_{(r,v_r)}) = 1$, do the following operations:

First Case: If the user is a member of role r and r has permission to read the file, i.e. tuple $\langle UR, u, (r, v_r), Enc^{IBE}_{u}(EK_{(r,v_r)}, SK_{(r,v_r)}), Sign^{IBS}_{SU}\rangle$ and $\langle PA, (r, v_r), (fn, op), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$ exist, download these two tuples from the cloud. Note that because of the write-time re-encryption policy, you need to download the *PA* tuple whose v_{fn} is consistent with tuple F, the same is true of the following. Then verify the signature of the tuple, if:

$Verify^{IBS}_{SU}(\langle UR, u, (r, v_r), Enc^{IBE}_{u}(EK_{(r,v_r)}, SK_{(r,v_r)})\rangle, Sign^{IBS}_{SU}) = 1$,

$Verify^{IBS}_{SU}(\langle PA, (r, v_r), (fn, op), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU\rangle, Sign^{IBS}_{SU}) = 1$, perform the following operations:

Step 1: The user uses his own decryption key EK_u to decrypt the decryption key and the signature key of the role r from the UR tuple: $(EK_{(r,v_r)}, SK_{(r,v_r)}) = Dec^{IBE}_{EK_u}(Enc^{IBE}_u(EK_{(r,v_r)}, SK_{(r,v_r)}))$.

Step 2: Use r's decryption key $EK_{(r,v_r)}$ to decrypt the symmetric key of the encrypted file from the PA tuple: $k = Dec^{IBE}_{EK_{(r,vr)}}(Enc^{IBE}_{(r,v_r)}(k))$.

Step 3: Use k to decrypt the encrypted file from the F tuple: $f = Dec^{Sym}_k(Enc^{Sym}_k(f))$.

Second Case: If the user is a member of the role r and r is a child role of the role r', r' has the permission to read the file fn, i.e. the following tuple exists:

$$\langle UR, u, (r, v_r), Enc^{IBE}_u(EK_{(r,v_r)}, SK_{(r,v_r)}), Sign^{IBS}_{SU}\rangle$$

$$\langle RH, (r, v_r), (r', v_{r'}), Enc^{IBE}_{(r,v_r)}(EK_{(r',v_{r'})}, SK_{(r',v_{r'})}), Sign^{IBS}_{SU}\rangle$$

$$\langle PA, (r', v_{r'}), (fn, op), v_{fn}, Enc^{IBE}_{(r',v_{r'})}(k), SU, Sign^{IBS}_{SU}\rangle$$

Then download the three tuples from the cloud and verify the signature. If the signature is valid, perform the following operations:

Step 1: The user uses his own decryption key EK_u to decrypt the decryption key and the signature key of the role r from the UR tuple: $(EK_{(r,v_r)}, SK_{(r,v_r)}) = Dec^{IBE}_{EK_u}(Enc^{IBE}_u(EK_{(r,v_r)}, SK_{(r,v_r)}))$.

Step 2: Use r's decryption key $EK_{(r,v_r)}$ to decrypt the decryption key and signature key of r' from the RH tuple:

$$(EK_{(r',v_{r'})}, SK_{(r',v_{r'})}) = Dec^{IBE}_{EK_{(r,vr)}}(Enc^{IBE}_{(r,v_r)}(EK_{(r',v_{r'})}, SK_{(r',v_{r'})}))$$

Step 3: Use r''s decryption key $EK_{(r',v_{r'})}$ to decrypt the symmetric key of the encrypted file from the PA tuple: $k = Dec^{IBE}_{EK_{(r',v_{r'})}}(Enc^{IBE}_{(r',v_{r'})}(k))$.

Step 4: Use k to decrypt the encrypted file from the F tuple: $f = Dec^{Sym}_k(Enc^{Sym}_k(f))$.

$write_u(fn,f)$: File writing algorithm. The user writes the file, that is, updates the file. The main steps are as follows:

First Case: If the user is a member of role r and r has permission to write the file fn. That is, tuple $\langle UR, u, (r, v_r), Enc^{IBE}_u(EK_{(r,v_r)}, SK_{(r,v_r)}), Sign^{IBS}_{SU}\rangle$ and $\langle PA, (r, v_r), (fn, RW), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$ exist. Then download these two tuples from the cloud and verify the signature. Note that due to the write-time re-encryption policy, the latest version of the file key is used here, i.e. download the largest version of the PA tuple in v_{fn}. The same is true of the following.

If $Verify^{IBS}_{SU}(\langle UR, u, (r, v_r), Enc^{IBE}_u(EK_{(r,v_r)}, SK_{(r,v_r)})\rangle, Sign^{IBS}_{SU}) = 1$ and $Verify^{IBS}_{SU}(\langle UR, (r, v_r), (fn, RW), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU\rangle, Sign^{IBS}_{SU}) = 1$, perform the following operations:

Step 1: The user uses his own decryption key EK_u to decrypt the role's decryption key and signature key from the *UR* tuple: $(EK_{(r,v_r)}, SK_{(r,v_r)}) = Dec^{IBE}_{EK_u}(Enc^{IBE}_u(EK_{(r,v_r)}, SK_{(r,v_r)}))$.

Step 2: User r's decryption key $EK_{(r,v_r)}$ to decrypt the symmetric key of the encrypted file from the *PA* tuple: $k = Dec^{IBE}_{EK_{(r,v_r)}}(Enc^{IBE}_{(r,v_r)}(k))$.

Step 3: Build the new tuple $\langle F, fn, v_{fn}, Enc^{Sym}_k(f'), (r, v_r), Sign^{IBS}_{(r,v_r)}\rangle$ and upload it to *R.M*.

Step 4: After *R.M* receives the tuple, it performs the following operations:

(1) Check that the tuple's format is correct. If the tuple's format is correct, then proceed to the next step.
(2) Verify the signature. If $Verify^{IBS}_{(r,v_r)}(\langle F, fn, v_{fn}, Enc^{Sym}_k(f'), (r, v_r)\rangle, Sign^{IBS}_{(r,v_r)}) = 1$, proceed to the next step.
(3) *R.M* checks if role r has permission to write the file *fn*. That is, check if tuple $\langle PA, (r, v_r), (fn, RW), v_{fn}, Enc^{IBE}_{(r,v_r)}(k), SU, Sign^{IBS}_{SU}\rangle$ exists. If the tuple exists and the signature is valid, then *R.M* uses the new tuple F instead of the old tuple.
(4) Delete all *PA* tuples associated with file *fn* and whose version numbers are less than v_{fn}.

Second Case: If the user is a member of the role r and r is a child role of the role r', r' has the permission to write the file *fn*, i.e. the following tuple exists:

$$\langle UR, u, (r, v_r), Enc^{IBE}_u(EK_{(r,v_r)}, SK_{(r,v_r)}), Sign^{IBS}_{SU}\rangle$$

$$\langle RH, (r, v_r), (r', v_{r'}), Enc^{IBE}_{(r,v_r)}(EK_{(r',v_{r'})}, SK_{(r',v_{r'})}), Sign^{IBS}_{SU}\rangle$$

$$\langle PA, (r', v_{r'}), (fn, RW), v_{fn}, Enc^{IBE}_{(r',v_{r'})}(k), SU, Sign^{IBS}_{SU}\rangle$$

Download the three tuples from the cloud and verify the signature. If the signature is valid, perform the following operations:

Step 1: The user uses his own decryption key EK_u to decrypt the role r's decryption key and signature key from the *UR* tuple: $(EK_{(r,v_r)}, SK_{(r,v_r)}) = Dec^{IBE}_{EK_u}(Enc^{IBE}_u(EK_{(r,v_r)}, SK_{(r,v_r)}))$.

Step 2: Use r's decryption key $EK_{(r,v_r)}$ to decrypt the decryption key and signature key of r' from the *RH* tuple:

$$(EK_{(r',v_{r'})}, SK_{(r',v_{r'})}) = Dec^{IBE}_{EK_{(r,v_r)}}(Enc^{IBE}_{(r,v_r)}(EK_{(r',v_{r'})}, SK_{(r',v_{r'})})).$$

Step 3: Use r''s decryption key $EK_{(r',v_{r'})}$ to decrypt the symmetric key of the encrypted file from the *PA* tuple: $k = Dec^{IBE}_{EK_{(r',v_{r'})}}(Enc^{IBE}_{(r',v_{r'})}(k))$.

Step 4: Build the new tuple $\langle F, fn, v_{fn}, Enc^{Sym}_k(f'), (r', v_{r'}), Sign^{IBS}_{(r',v_{r'})}\rangle$ and upload it to *R.M*.

Step 5: After *R.M* receives the tuple, it performs the following operations:

(1) Check that the tuple's format is correct. If the tuple's format is correct, then proceed to the next step.
(2) Verify the signature. If $Verify^{IBS}_{(r,v_r)}(\langle F, fn, v_{fn}, Enc^{Sym}_{k}(f'), (r', v_{r'})\rangle, Sign^{IBS}_{(r',v_{r'})}) = 1$, proceed to the next step.
(3) *R.M* checks if role r has permission to write the file *fn*. That is, check if the following tuples exist:

$$\langle RH, (r, v_r), (r', v_{r'}), Enc^{IBE}_{(r,v_r)}(EK_{(r',v_{r'})}, SK_{(r',v_{r'})}), Sign^{IBS}_{SU}\rangle$$

$$\langle PA, (r', v_{r'}), (fn, op), v_{fn}, Enc^{IBE}_{(r',v_{r'})}(k), SU, Sign^{IBS}_{SU}\rangle$$

If the tuples exist and the signature is valid, then *R.M* uses the new tuple F instead of the old tuple.

Delete all *PA* tuples associated with file *fn* and whose version numbers are less than v_{fn}.

3 Conclusions

This paper combines identity-based cryptosystems and role-based access control models ($RBAC_1$ model), and built an RBAC scheme based on identity cryptosystem in cloud storage. This paper first describes the application scenario and entity composition of the scheme; then gives the formal definition of the scheme; then describes the key technologies of the scheme, and introduces the four tuples used to describe the access control policy in detail and the designed optimization method in order to improve system efficiency-hybrid encryption policy and write-time re-encryption policy; this paper also gives a detailed description of the scheme, specific to each step of each operation; finally analyzes the scheme to prove that the scheme is correct, access control protected and secure.

Acknowledgement. This work is supported, in part, by the National Natural Science Foundation of China under grant No. 61872069, in part, by the Fundamental Research Funds for the Central Universities (N171704005), in part, by the Shenyang Science and Technology Plan Projects (18-013-0-01).

References

1. Peng, S., Zhou, F., Jian, X., Zifeng, X.: Comments on "identity-based distributed provable data possession in multicloud storage". IEEE Trans. Serv. Comput. **9**(6), 996–998 (2016)
2. Xu, J., Wei, L., Zhang, Y., Wang, A., Zhou, F., Gao, C.: Dynamic fully homomorphic encryption-based Merkle Tree for lightweight streaming authenticated data structures. J. Netw. Comput. Appl. **107**, 113–124 (2018)

3. Jung, Y., Chung. M.: Adaptive security management model in the cloud computing environment. In: The International Conference on Advanced Communication Technology, pp. 1664–1669. IEEE (2010)
4. Wang, X.W., Zhao, Y.M.: A task-role-based access control model for cloud computing. Comput. Eng. **38**(24), 9–13 (2012)
5. Danwei, C., Xiuli, H., Xunyi, R.: Access control of cloud service based on UCON. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom 2009. LNCS, vol. 5931, pp. 559–564. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10665-1_52
6. Krautsevich, L., Lazouski, A., Martinelli, F., et al.: Risk-aware usage decision making in highly dynamic systems. In: Fifth International Conference on Internet Monitoring and Protection, pp. 29–34. IEEE (2010)
7. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_27
8. Goyal, V., Pandey, O., Sahai, A., et al.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM Conference on Computer and Communications Security, pp. 89–98. ACM (2006)
9. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: CCS 07 ACM Conference on Computer & Communications Security, pp. 195–203 (2007)
10. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, pp. 321–334. IEEE Computer Society (2007)
11. Sun, G.Z., Yu, D., Yun, L.I.: CP-ABE based data access control for cloud storage. J. Commun. **32**(7), 146–152 (2011)
12. Jung, T., Li, X.Y., Wan, Z., et al.: Privacy preserving cloud data access with multi-authorities. In: 2013 Proceedings IEEE INFOCOM, pp. 2625–2633. IEEE (2013)
13. Ruj, S., Stojmenovic, M., Nayak, A.: Privacy preserving access control with authentication for securing data in clouds. In: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 556–563. IEEE (2012)
14. Yu, S., Wang, C., Ren, K., et al.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: 2010 Proceedings IEEE INFOCOM, pp. 1–9. IEEE (2010)
15. Hur, J., Dong, K.N.: Attribute-based access control with efficient revocation in data outsourcing systems. IEEE Trans. Parallel Distrib. Syst. **22**(7), 1214–1221 (2011)
16. Chen, D.W., Shao, J., Fan, X.W., et al.: MAH ABE based privacy access control in cloud computing. Acta Electron. Sin. **42**(4), 821–827 (2014)