Anonymous Credential-Based Access Control Scheme for Clouds

Xuanxia Yao, University of Science and Technology Beijing
Hong Liu, Run Technologies Co., Ltd., Beijing
Huansheng Ning, University of Science and Technology Beijing
Laurence T. Yang, St. Francis Xavier University, Canada
Yang Xiang, Deakin University, Australia

A lightweight ciphertext sharing scheme uses an anonymous authorization credential to simplify access control, ensure users' anonymity, and support decryption key reconstruction.

With the development of cloud computing, more data is being stored in the cloud. Because cloud users might not want to disclose their information to cloud servers, security and privacy protection are basic requirements of cloud storage. Sensitive data is usually stored as ciphertext, but must be sharable with authorized users and certain services. Designing ciphertext control mechanisms to achieve access control in cloud storage is therefore challenging.

Proposed ciphertext control mechanisms for data access control include access control list (ACL),[1] attribute-based encryption (ABE),[2–4] and proxy reencryption-based solutions.[5] ACL-based solutions are inflexible and have poor scalability. ABE-based solutions can overcome these shortcomings, achieving both security and privacy requirements,[6,7] because they allow any user to access the ciphertext as long as the user's attributes match the encryption attributes. However, these solutions use complex bilinear mapping, which brings a high compute overhead. In addition, authorization management is inflexible and the open attributes could compromise the user's privacy. Proxy reencryption-based solutions support access control through ciphertext transformation. Although they're flexible, high overhead remains an issue because of the bilinear mapping and public key transmission.

 2325-6095/15/$31.00

We've designed a lightweight ciphertext sharing scheme that uses an anonymous authorization credential to simplify access control, ensure users' anonymity, and support decryption key reconstruction. We adopt a hierarchical key structure to realize fine-grained key management and developed a flexible and lightweight key reconstruction solution based on the Lagrange interpolation function. By involving only simple arithmetic and symmetric and asymmetric cryptographic algorithm operations, our solution overcomes the high overhead caused by bilinear mapping in most existing schemes. Performance evaluation and analysis show that the proposed scheme has advantages over existing approaches.

Motivation

At present, the basic idea of restricting access to encrypted data is to limit users' decryption power by disclosing the data encryption key only to authorized users. This process involves three main issues: key management, authorization management, and privacy preservation.

For key management, ACL-based solutions use symmetric and asymmetric algorithms to distribute the decryption key to authorized users, so the overhead grows with the ACL's length. The ABE scheme embeds an access control policy into an encryption algorithm to allow a user to decrypt the ciphertext if the user meets the required attributes. Because key management is integrated into access control policies, ABE solutions can avoid frequent key distribution during ciphertext access control. Proxy reencryption solutions realize key distribution by transforming the ciphertext of the encryption key to be encrypted by the authorized user's public key. Although this scheme might seem flexible, it has a high computation overhead.

For authorization management, ACL- and proxy-reencryption-based solutions are relatively simple. ABE solutions for updating the access control policy are complicated, and are neither flexible nor practical.

For privacy preservation, ACL- and proxy-reencryption-based solutions can't provide privacy preservation for users, because they need to know the user's public key. ABE solutions also risk data leakage, because fine-grained access control requires additional attributes.[3]

f_i	CF_i	CK_i	MF_i	MD_i

FIGURE 1. Formatting for files stored in the cloud. Files have five parts: file identification (f_i), cyphertext of the encrypted file (CF_i), cyphertext of the encrypted key (CK_i), the encrypted file's secret share (MF_i), and the file's integrity check code (MD_i).

Recently, several credential-based access control schemes have been proposed.[8–10] Some are designed for anonymous access control on plaintext and aren't suitable for ciphertext access control. Although some others are designed for anonymous access control on ciphertext, the decryption key is only distributed by the credential and isn't flexible enough in the key updating. A lightweight, flexible ciphertext access control scheme with privacy preservation is clearly needed.

System Assumptions and Notation

The system model involves three types of participants: cloud server, data owner, and data sharer.

A cloud server provides data storage services and enforces access control on the stored data according to authorization credentials. A cloud server is considered semitrusted, meaning it can perform access control on the encrypted files faithfully and keep authorization-related data secret, but it's curious about the plaintext of the data stored in it. A cloud server should help the data owner manage authorization. To support authorization management, the cloud server needs to create an authorization credential list for each registered user and add an item for each authorization credential to point to the revoked files' names list.

A data owner is a registered cloud user who stores files in the cloud in the format shown in Figure 1 (see Table 1 for a listing of the notations and symbols used in our scheme), and can share the files with other users by issuing authorization credentials to them. To avoid issuing more than one authorization credential to a data sharer, a data owner records and indexes all valid authorization credentials. A data owner can revoke the authorization from a data sharer without notification.

Data sharers can access encrypted files listed in the authorized files list (I_1) of their authoriza-

Table 1. Notations and symbols.

Notation	Description
CF_i	Ciphertext of f_i, $CF_i = E(KF_i, F_i)$
CK_i	Ciphertext of f_i's encryption key, $CK_i = E(KP_i, KF_i)$
$D(k, C)$	Decrypt ciphertext C using key k
$E(k, m)$	Encrypt message m using key k
f_i	Full name of file i
F_i	Contents of f_i, if f_i is a directory, $F_i = f_i$
Hash()	Hash function
HMAC(k, m)	Keyed-hash message authentication code (HMAC) function, based on key k and message m
ID_i	Identifier of i
KF_i	Encryption key based on the symmetric cryptograph for f_i
KO_i	Secret key of data owner i
KP_i	Parent directory key of fi
MD_i	Digest of CF_i, $MD_i = Hash(CF_i)$
MF_i	Key sharer of CF_i, $MF_i = gi(Hash(CF_i))$
P	A big prime; its length should not be less than the required length of the symmetric key
PK_i	Public key of i
Sig(k, m)	Signature of message m using key k
SK_i	Private key of i

tion credentials and their mask values by showing proof to the cloud server that they're the owner of the authorization credentials. Based on the encrypted file, its mask value, and its file binding code, a data sharer can reconstruct the file key of the authorized file. Data sharers should also keep the sensitive data derived from the authorization credential secret.

System initialization proceeds as follows:

- Data owners register with the cloud server.
- The cloud server allocates storage space for the registered users.
- A pair of public/private keys are generated for each participant.
- Participants can get each other's public keys.
- All participants can perform symmetrical and asymmetrical encryption/signature algorithms.
- Data files in the cloud are organized using a directory structure for each user.

The hash function and keyed-hash message authentication code (HMAC) function are noncollision functions or security in a cryptographic sense.

Key Management

The hierarchical key structure is popular in cloud storage.[11] Generally, the user-oriented key, or top-level key, is used to decrypt a file and is usually managed by a public key infrastructure (PKI) or trusted third party (TTP), or is integrated into a set of attributes or access control policies. The file-oriented key, or non-top-level key, is used to encrypt the file and is usually stored in the form of ciphertexts with the encrypted file.

Here, we use the file system's directory structure to organize and manage keys, since it's consistent with the file system and makes it easy to realize fine-grained key management (one file, one key). Accordingly, there are three types of keys—the data owner's secret key, the directory key, and the data file key.

Key Generation

For a data owner A, its secret key KO_A can be randomly generated according to the security policy and must be secretly kept by itself. As the top-level key, it's used to encrypt the key of each file in its root directory.

Because a directory and a data file can both be considered files for practical purposes, the directory

and data file keys can be generated in the same way. For fine-grained key management, and given that changing a file (especially its name) means changing the authorization, the file key, shown in Equation 1, is unique for each file and changes with the file's name:

$$KF_i = HMAC(r_i, f_i), \quad (1)$$

where r_i is chosen from [1, p] randomly for file f_i.

Note that the functions of the directory key and the data file key are different. The former is used to encrypt the keys for all of the files in the directory, whereas the latter is used to encrypt only the data file itself.

Key Distribution

Key distribution occurs when the data owner wants to grant access privileges to a data user. To allow only the authorized data user to decrypt the authorized file, we use Shamir's Secret Sharing to distribute the decryption key.[12] We model the key distribution using a (2, 2) Shamir's Secret Sharing problem. For each file, a 1-degree Lagrange interpolation function over the finite field Z_p should be constructed using Equation 2, where $a, b \in Z_p$:

$$g(x) = (ax + b) \bmod p. \quad (2)$$

To distribute f_i's key only to its authorized user and simplify key management, the interpolation function of file f_i is required to pass point (0, FBC_i). Here, FBC_i is the file binding code of f_i and is calculated using Equation 3:

$$FBC_i = HMAC(KO_A, f_i) \bmod p. \quad (3)$$

For security, FBC_i should be encrypted with the credential owner's public key, and transmitted using the authorization credential. Meanwhile, CF_i is considered to be a participant in the sharer of the key; its secret share $MF_i = g(Hash(CF_i) \bmod p)$ is stored with it in the cloud.

Thus, we can describe f_i's key distribution function using Equation 4:

$$g_i(x) = (KF_i \times x + FBC_i) \bmod p. \quad (4)$$

Only the credential owner can obtain the two points ($Hash(CF_i)$, MF_i) and (0, FBC_i) on f_i's interpolation function, and reconstruct f_i's key using Equation 5:

$$KF_i = (Hash(CF_i))^{-1} \times (MF_i - FBC_i) \bmod p. \quad (5)$$

I_1: Authorized files list (f_1, f_2, ... f_n)
I_2: Validity period
I_3: Credential number (*CN*)
I_4: Credential issuer's or data owner's ID
I_5: Credential issuer's or data owner's signature
I_6: Verification code (*VC*)
I_7: Binding code list (M_0, M_1, M_2, ... M_n)

FIGURE 2. Anonymous authorization credential, where I_1, I_2, I_3, I_4, and I_5 are the credential's basic information, which is requisite for the credential owner and the credential verifier; I_6 is known only to the cloud server; and I_7 is known only to the credential owner.

Because different files have different interpolations and file binding codes, unauthorized users (even if they've stolen an authorized user's credential) can't get the two points for file key reconstruction.

There are two ways to obtain the file key. In one case, where the requested file is in the authorized file listed, the file key can be reconstructed according to the interpolation function. In the other case, where the required file isn't in the authorized file listed but its parent or ancestor directory is, the parent or ancestor directory key should first be reconstructed using the interpolation function, and then the file key can be obtained using stepwise decryption.

Key Updating

Key updating usually occurs when the current key expires, the file name changes, the file changes, or the data owner wants to revoke the access privilege for a file from all data users. Most of the operations are similar to key generation. The main difference is that key updating often involves some reencryption. Due to length limitations, we don't discuss this here.

Authorization Credential

We introduce an anonymous authorization credential to express the authorization information and distribute a key sharer of the authorized file to the credential owner. This credential has seven items, denoted I_1, I_2, I_3, I_4, I_5, I_6, and I_7, as Figure 2 illustrates.

Item I_1 expresses the authorization objects as an authorized files list. It includes the full names of the files whose access privileges are granted to the credential owners.

Items I_2, I_3, and I_4 are jointly used to describe the credential information and indicate the validity period, credential number, and the credential issuer's ID, respectively. Items I_3 and I_4 should be unique in the same system.

Item I_5 is the credential issuer A's (that is, the data owner) signature on the authorization and credential information. It helps the cloud server C and the authorized user B (that is, the data sharer or credential owner) verify the credential's validity. I_5 is determined by following two steps:

1. Compute the digest of the basic information of the credential, HC = Hash($I_1||I_2||I_3||I_4$).
2. Add signature to HC and get I_5 = Sig(SKA, HC).

Item I_6 is the verification code (VC), which is mainly used by cloud server C to check whether the credential is used by its owner or not. We determine I_6 by following three steps:

1. Compute the credential binding code (CBC) using Equation 6, which is used to bind the credential with its owner and avoid being stolen by another user:

$$\text{CBC} = \text{HMAC}(\text{KO}_A, \text{CN}||\text{ID}_B), \quad (6)$$

   where CN is the credential number and ID_B is the data sharer's identity.
2. Compute the user verification code (UVC) using Equation 7:

$$\text{UVC} = \text{HMAC}(\text{CBC}, \text{CN}). \quad (7)$$

3. Calculate $I_6 = E(\text{PK}_C, \text{UVC})$, where PK_C is the public key of cloud server C.

Item I_7 is the binding code list, which is used by the credential owner B to verify the credential's integrity, and to extract the credential and file binding codes. The CBC is used to prove the user is the credential owner, and the FBC is used as a secret share to restore the file key. To prevent unauthorized users from stealing the credential and restoring the file key, all binding codes should be transmitted in ciphertext. We determine I_7 using the following three steps:

1. Encrypt CBC using PK_B to get M_0, $M_0 = E(\text{PK}_B, \text{CBC})$.
2. For each authorized file f_i in I_1, compute FBC_i using Equation 3 and encrypt it using PK_B to get M_i, $M_i = E(\text{PK}_B, \text{FBC}_i)$, $i \in [1, n]$.
3. Get $I_7 = M_0||M_1||, \ldots, ||M_n$, $i \in [1, n]$.

The authorization credential contains only the first five items, which have nothing to do with the credential owner's identity. Items I_6 and I_7 are designed for the cloud server and the credential owner, respectively, and are related to the identity of credential owner B by the CBC, which is used to bind B and the credential by $\text{HMAC}(\text{KO}_A, \text{CN}||\text{ID}_B)$. Obviously, CBC can hide ID_B from anyone. Because none of credential owner B's identity information is exposed, the authorization credential can be regarded as anonymous.

Scheme Description

The proposed ciphertext access control scheme includes four function modules: file management, authorization, access control, and authorization revoking. In essence, the last three function modules are elements of authorization management. Figures 3 and 4 show diagrams of file management and authorization management, respectively.

File Management

Similar to existing cloud storage systems, registered data owners can log into their accounts to manage their files. The main differences are the processes for creating and modifying files. Because a data owner A can derive a file key from its secret key, the data owner uses the parent directory key KP_i of f_i to encrypt the key of a new file in it.

Creating a file. Creating a file means creating a directory or uploading a data file to the data owner's account in the cloud. For a registered user A (that is, the data owner), constructing a CFR for f_i requires performing the following eight steps:

1. Choose a random number r_i from [1, p], and calculate its key according to Equation 1, $\text{KF}_i = \text{HMAC}(r_i, f_i)$.
2. Encrypt KF_i with KP_i, $\text{CK}_i = E(\text{KP}_i, \text{KF}_i)$.
3. If the new file is a directory, let F_i be f_i. Encrypt F_i with KF_i and get $\text{CF}_i = E(\text{KF}_i, F_i)$.
4. Compute $\text{FBC}_i = \text{HMAC}(\text{KO}_A, f_i)$.
5. Construct a linear function $g_i(x) = \text{KF}_i \times x + \text{FBC}_i$ on a finite field Z_p for f_i and compute the secret sharer MF_i for CF_i, $\text{MF}_i = \text{KF}_i \times \text{Hash}(\text{CF}_i) + \text{FBC}_i$.
6. Compute $\text{MD}_i = \text{Hash}(\text{CF}_i)$.
7. Compute H_i = Hash(Creating$||f_i||\text{CF}_i||\text{CK}_i$

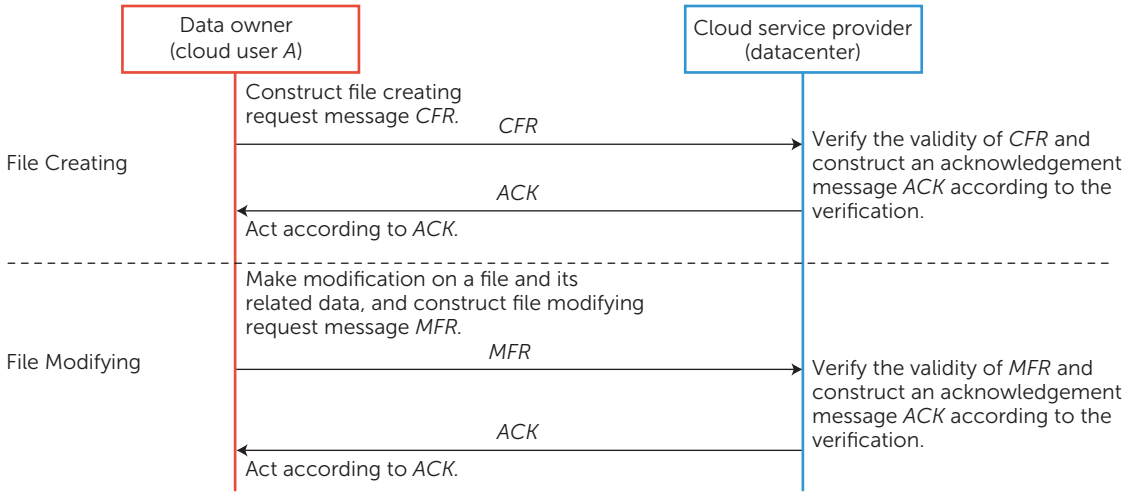

FIGURE 3. File management. (a) In the file creation stage, data user A constructs a create file request (CFR) according to the format shown in Figure 1. (b) In the file modification stage, a modify file request (MFR) is constructed similarly to the CFR.

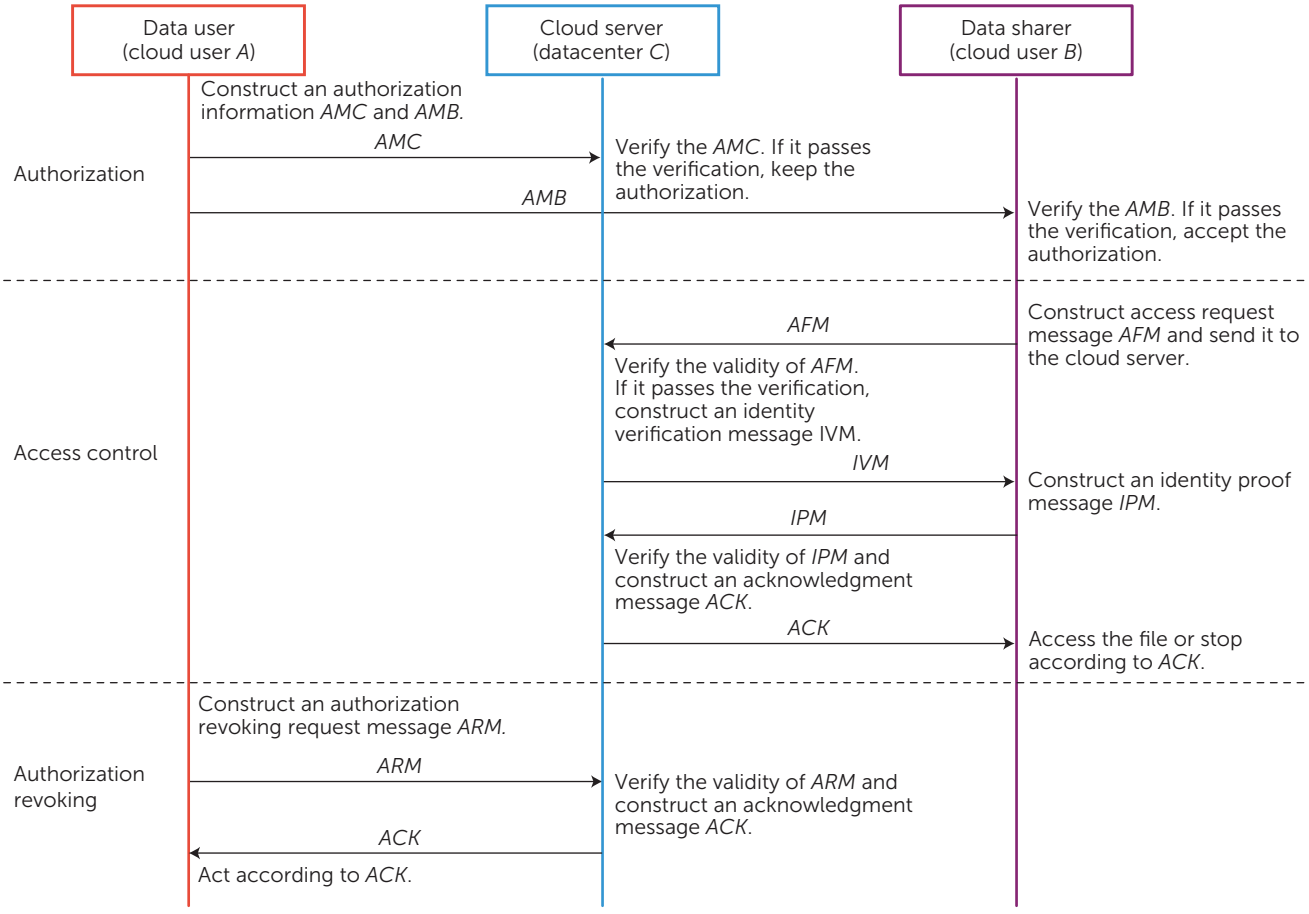

FIGURE 4. Authorization management. (a) Authorization information should be constructed according to two cases, for the cloud server and data sharer, respectively. (b) Access control involves only the cloud server and data sharer. The cloud server uses an access request message (AFM), identity verification message (IVM), and identity proof message (IPM) to access the encrypted file. The data sharer uses an acknowledgment message (ACK) to reconstruct the file key. (c) Authorization revocation is launched by the data owner and accomplished by the cloud server.

$||MF_i||MD_i$), and add a signature to it, SH_i = $Sig(SK_A, H_i)$.
8. Let CFR = $Creating||f_i||CF_i||CK_i||MF_i||MD_i||SH_i||ID_A$, and send it to the cloud server.

After receiving the CFR, the cloud server verifies it using PK_A. If it passes verification, f_i will be stored in A's cloud account in the format shown in Figure 1. Meanwhile, the cloud server will send a "success" ACK to A; otherwise, it will send an "error" ACK to A.

Modifying a file. When a data owner A wants to modify its file f_i stored in the cloud, it performs the following six steps.

1. Make modification on f_i or F_i to obtain f_i' or F_i'.
2. If $f_i \neq f_i'$ or A wants to update KF_i, choose a random number r_i' from [1, p], and compute $KF_i' = HMAC(r_i', f_i')$; otherwise, let $KF_i' = KF_i$.
3. Perform the operations in "creating a file" from steps 2 to 6 to get CK_i', CF_i', FBC_i', MF_i', and MD_i', respectively.
4. Compute $H_i' = Hash(Modifying||f_i||f_i'||CF_i'||CK_i'||MF_i'||MD_i')$.
5. Make a signature on H_i' and get $SH_i' = Sig(SK_A, H_i')$.
6. Let MFR = $Modifying||f_i||f_i'||CF_i'||CK_i'||MF_i'||MD_i'||SH_i'||ID_A$, and send it to the cloud server.

After receiving the MFR, the cloud server will verify it using PK_A. If the file passes verification, the cloud server checks whether f_i is in A's account or not. If it is, it deletes f_i and stores f_i' in the format shown in Figure 1 in the proper location, and sends a "success" ACK to data owner A; otherwise, it sends "error" to A.

Authorization

Authorization includes both whole and partial authorization. Whole authorization is used when there's no valid authorization credential for the data user. In this case, the whole authorization credential should be constructed on the authorized files. The authorization information for the cloud server (AMC) is set to "$I_1||I_2||I_3||I_4||I_5||I_6$," and the authorization information for the data user B (AMB) is set to "$I_1||I_2||I_3||I_4||I_5||I_7$."

Partial authorization is used when there's a valid authorization credential for data user B, and data owner A wants to add some authorization to it. Here, the data owner just needs to tell the cloud server and the data sharer the added authorization information instead of the whole credential. For this purpose, the file list granted to B should be constructed and denoted by *Pf*. Accordingly, the file binding code for each file in *Pf* should be computed and encrypted by PK_B to get the FBC list, which is denoted *Mf*. For the cloud server, AMC is set to "$Adding||ID_A||CN|Pf||Sig(SK_A, Hash(Adding||ID_A||CN||Pf))$," and for B, AMB is set to "$Adding||ID_A||Pf||Mf||Sig(SK_A, Hash(Adding||ID_A||Pf||Mf))$."

Access Control

Access control includes two aspects. One is making access control on the ciphertext, and the other is file key reconstruction. For the former, five steps are needed.

Data user B performs step 1, which includes:

- Decrypt M_0 in I_7 of the authorization credential to get CBC.
- Construct an access request "$f_i||II_3||I_4||$ $E(PK_C,$ CBC)" and send it to cloud server C.

Cloud server C performs step 2, which involves checking whether the credential is valid and f_i is an authorized file of the authorization credential. If yes, it decrypts $E(PK_C, CBC)$ to compute UVC = HMAC(CBC, CN). If UVC = I_3, generate a random number r from [1, p] and let ACK be $E(CBC, r)$; otherwise, let ACK be "error."

In step 3, data user B decrypts $E(CBC, r)$ with CBC, and encrypts $(r + 1)$ with CBC and sends it to C.

In step 4, cloud server C checks whether $D(CBC, E(CBC, (r + 1))) = (r + 1)$ or not. If not, let ACK be "error"; if so, there are two cases:

- ACK = $E(CBC, CF_i||MF_i||MD_i)$, if f_i is in I_1.
- ACK = $E(CBC, CF_j||MF_j|| \ldots CK_p||CF_i||CK_i||MD_i)$, if f_i is not in I_1, but its ancestor directory f_j is in I_1. Here, " $\ldots CK_p$" denotes that all CK_s are in the path of f_j to f_p.

In step 5, data user B reconstructs the file key according to ACK. If the ACK is "error," stop the reconstruction; otherwise, decrypt it and verify the integrity of CF_i. If it does not pass verification, stop; otherwise, reconstruct the file key as follows:

- if f_i is in I_1, decrypt M_i with CBC to get FBC_i; compute $KF_i = (Hash(CF_i))^{-1} \times (MF_i - FBC_i)$;
- otherwise, decrypt M_j with CBC to get FBC_j; compute $KF_j = (Hash(CF_j))^{-1} \times (MF_j - FBC_j)$, and derive KF_i from KF_j and the CK of each of its ancestor stepwise.

After obtaining the decryption key of the requested file, data user B can decrypt its ciphertext CF_i with it and get its plaintext F_i to read.

Authorization Revoking

Authorization revoking is launched by the data owner and accomplished through collaboration of the data owner and cloud server. It includes three steps:

1. Data owner A constructs a revoking file list denoted by *Rf*, and makes a signature on Hash(Revoking||ID_A||CN||*Rf*) to get SR = Sig(SK_A, Hash(Revoking||ID_A||CN||*Rf*)). It then sends "Revoking||ID_A||CN||*Rf*||SR" to cloud server C.
2. Cloud server C verifies SR with PK_A. If SR passes verification and the credential with CN is in A's authorization credential list, for each file in *Rf,* if it is in I_1, delete it from both *Rf* and I_1. If I_1 is empty, remove the credential from the credential list, and let ACK be Sig(SK_C, SR); otherwise, let ACK be "error." And then sends ACK to data owner A.
3. If ACK isn't "error," data owner A verifies Sig(SK_C, SR) with PK_C. If it passes verification, delete the file names in *Rf* from I_1. If I_1 is empty, delete the credential.

The process of authorization revoking requires the data owner and the cloud server to authenticate each other so as to keep the consistency of the credential in both sides.

Security and Performance Evaluation

To evaluate the proposed scheme, we use methods described elsewhere to analyze security and performance.[13,14] We also analyze the overheads, considering the aspects of key management and authorization management.

The scheme's security is guaranteed by linear secret sharing and symmetric and asymmetric theory. We assume that all algorithms and functions are secure.

For confidentiality, we encrypt the data in the cloud using a security symmetric cryptographic algorithm, which makes the data secret from the cloud server. The key distribution is based on the idea of linear secret sharing, and only the authorized user can obtain secret sharers, which also keeps it secret from unauthorized users. So, the encryption operation is limited to authorized users and the data owner.

The hash function and the signature based on the asymmetric algorithm are jointly used to guarantee data integrity. Both the cloud server and the authorized user can check data integrity as necessary.

Authenticity is guaranteed through authentication, which is realized by a signature based on an asymmetric algorithm or a symmetric cryptographic algorithm. For instance, both a signature and encryption are used in access control; the former helps the cloud server verify the credential's authenticity, and the latter helps it guarantee that the user is the authorization credential owner.

Fine-Grained Access Control

Access control granularity can be refined to a basic data unit and a basic access entity. In this scheme, the data owner can specify who can access which file (such as a data block, data file, or directory) during a certain valid period based on a specific authorization credential. The authorization credential is bounded with its data owner anonymously, and also stays consistent with the data users.

Scalability

In this scheme, the size of the encrypted data and the length of the parameters are relatively independent of the number of data users. Adding a data user means issuing it an authorization credential, and there's no change to the data in the cloud, and it doesn't involve the cloud server or affect other data users either. Thus, adding a data user doesn't cause an obvious performance decline or cost within the system.

Overhead

For simplicity, we measure computational overhead using an approach in which one bilinear pairing is about 20 point scalar multiplications, and one modular exponential operation is two point scalar multiplications.[15] Because the overheads for symmetric cryptography and hash/HMAC are much less than those for public key operations, we consider only the public key operations.

Key Management

In this scheme, distributing a file key involves 1 elliptic curve cryptography (ECC) encryption and decryption. Key updating doesn't affect authorization and the FBC, and only symmetric encryption and arithmetical operations are needed. Assume we use the elliptic curve integrated encryption scheme (ECIES) to perform ECC encryption and decryption, and we need only three point scalar multiplications.

In traditional ABE schemes, key distribution needs one decryption involving one more bilinear pairing. Key updating involves one encryption operation. There are at least two bilinear pairings, which are about 40 point scalar multiplications. It can be seen that the cost for our key management is much lower than that of other ABE schemes.

Table 2. Performance description.

Property	Description	Technology method
Security	Confidentiality	Encryption based on symmetric cryptographic algorithm
	Integrity	Hash, HMAC
	Authenticity	Authentication based on elliptic curve cryptography (ECC)
	Key distribution	Authorization credential and secret sharing based on the Lagrange interpolation function
Fine-grained access control	Object granularity	From a data block to a directory
	Authorized entity	A user or user group
Dynamic	Data user's variability	Easy to add or delete a data user
Scalability	Ability to increase the number of data users without decreasing the performance	Adding a new data user involves issuing an authorization credential to it, which does not affect the data and other data users.
Accountability	Ability to revoke the anonymity	The collaboration of the cloud server and the data owner
Overhead	Key management	Three point scalar multiplications for a file
	Authorization management	At most 17 point scalar multiplications

Authorization management includes both granting and revoking authorizations. In this scheme, authorization granting aims to issue an authorization credential. One ECC-based encryption and one signature are needed for a credential and $(n + 1)$ ECC-based encryptions are needed for n files in an authorization credential. To revoke an authorization, if the authorization credential is for a single user, only one ECC-based signature is needed. If the authorization credential is for a group of users and the data owner wants to revoke some members' privilege, the data owner will revoke one credential and issue one credential; only one ECC-based signature and $(n + 1)$ ECC-based encryptions are needed. Assuming that $n \leqq 15$ and ECDSA is used for signature and ECIES is used for encryption, one signature and $(n + 2) \leqq 17$ encryption are needed, which is about 35 point scalar multiplications.

In typical ABE schemes, an access control policy is usually associated with the private key or ciphertext. Authorization is embedded in the encryption process, which needs at least one bilinear pairing mapping. Updating the access policy requires reencrypting the changed attribute set. Even the "lazy" reencryption strategy can be postponed; the expensive reencryption (at least one bilinear pairing mapping) is inevitable. Authorization revocation also means that changing and reencrypting attributes are unavoidable. Thus, at least three bilinear pairing mappings are needed, which is equivalent to 60 point scalar multiplications.

It turns out that the overhead of authorization management is much lower than that in other ABE schemes. Additionally, the proposed scheme provides partial revoking, which realizes flexible authorization management. Table 2 summarizes the performance analysis.

The proposed access control scheme is anonymous, lightweight, fine-grained, and scalable, but it's designed only for ciphertext access control and can't meet the access control requirements of other cloud applications. In the future, we'll try to balance the costs on the user side and the expense of using cloud resources to develop a practical service access control scheme.

Acknowledgments

This work was supported by National Natural Science Foundation of China under grant 61471035. It was jointly supported by the Fundamental Research Funds for the Chinese Central Universities under grant 06105031. This work was also supported by Science and Technology Foundation of Beijing (Z141100002714003). Xuanxia Yao is the corresponding author.

References

1. V. Kher and Y. Kim, "Securing Distributed Storage: Challenges, Techniques, and Systems," *Proc. ACM Workshop Storage Security and Survivability* (StorageSS 05), 2005, pp. 9–25.
2. S. Yu et al., "Achieving Secure Scalable and Fine-Grained Data Access on Cloud Computing,"

Proc. 29th Conf. Information Comm. (INFORM 10), 2010, pp. 534–542.
3. A. Lewko and B. Waters, "New Techniques for Dual System Encryption and Fully Secure HIBE with Short Ciphertexts," *Proc. 7th Theory of Cryptography Conf.* (TCC 10), 2010, pp. 455–479.
4. V. Goyal et al., "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data," *Proc. 13th ACM Conf. Computer and Comm. Security* (CCS 06), 2006, pp. 89–98.
5. Z. Yan, X. Li, and R. Kantola, "Controlling Cloud Data Access Based on Reputation," *Mobile Networks and Applications*, Mar. 2015; doi: 10.1007/s11036-015-0591-6.
6. H.T. Dinh et al., "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches," *Wireless Comm. and Mobile Computing,* vol.13, no.18, 2013, pp. 1587–1611.
7. H. Liu et al., "Shared Authority Based Privacy-Preserving Authentication Protocol in Cloud Computing," *IEEE Trans. Parallel and Distributed Systems*, vol. 26, no. 1, 2015, pp. 241–251.
8. A. Ahmad, M.M. Hassan, and A. Aziz, "A Multi-Token Authorization Strategy for Secure Mobile Cloud Computing," *Proc. 2nd IEEE Int'l Conf. Mobile Cloud Computing, Services, and Eng.*, 2014, pp. 136–141.
9. N.M. Gonzalez et al., "A Framework for Authentication and Authorization Credentials in Cloud Computing," *Proc. 12th IEEE Int'l Conf. Trust, Security and Privacy in Computing and Comm.* (TRUSTCOM 13), 2013, pp. 509–516.
10. D. Jana and D. Bandyopadhyay, "Management of Identity and Credentials in Mobile Cloud Environment," *Proc. Int'l Conf. Advanced Computer Science and Information Systems* (ICACSIS 13), 2013, pp. 113–118.
11. M.J. Atallah et al., "Dynamic and Efficient Key Management for Access Hierarchies," *ACM Trans. Information and System Security* (TISSEC), vol.12, no.3, 2009, article no. 18.
12. I.N. Bozkurt, K. Kaya, and A.A. Selçuk, "Practical Threshold Signatures with Linear Secret Sharing Schemes," *Progress in Cryptology—AFRICACRYPT 2009*, LNCS 5580, Springer, 2009, pp. 167–178.
13. A.N. Khan et al., "Towards Secure Mobile Cloud Computing: A Survey," *Future Generation Computer Systems*, vol. 29, no.5, 2013, pp. 1278–1299.
14. S.S.M. Chow et al., "Cryptography and Security-Dynamic Secure Cloud Storage with Provenance," *Cryptography and Security: From Theory to Applications*, LNCS 6805, Springer, 2012, pp. 442–464.
15. V.G. Martínez, L.H. Encinas, and C.S. Ávila, "A Survey of the Elliptic Curve Integrated Encryption Scheme," *J. Computer Science and Eng.*, vol. 2, no. 2, 2010, pp. 7–13.

XUANXIA YAO *is an associate professor in the School of Computer and Communication Engineering at the University of Science and Technology Beijing. Her research interests include network security, the Internet of Things, and cloud computing. Yao has a PhD in computer science from the University of Science and Technology Beijing, China. She's a member of the China Computer Federation. Contact her at yaoxuanxia@163.com.*

HONG LIU *is a research fellow at the Engineering Laboratory, Run Technologies Co., Ltd., Beijing, where she focuses on the security and privacy issues in RFID, vehicle-to-grid networks, and the Internet of Things. Her research interests include authentication protocol design, and security formal modeling and analysis. Liu has a PhD in circuits and systems from the School of Electronic and Information Engineering, Beihang University, China. She's a member of IEEE. Contact her at liuhongler@ieee.org.*

HUANSHENG NING *is a professor in the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. His current research interests include the Internet of Things, aviation security, electromagnetic sensing, and computing. Ning has a PhD in information and communication engineering from Beihang University. He's a senior member of IEEE. Contact him at ninghuansheng@ustb.edu.cn.*

LAURENCE T. YANG *is a professor in the Department of Computer Science at St. Francis Xavier University, Canada. His research interests include parallel and distributed computing and embedded and ubiquitous/pervasive computing. Yang has a PhD in computer science from the University of Victoria, Canada. He's a member of IEEE. Contact him at ltyang@stfx.ca.*

YANG XIANG *is the director of the Network Security and Computing Lab (NSCLab). His research interests include network and system security, distributed systems, and networking, and he's currently leading his team in developing active defense systems against large-scale distributed network attacks. Xiang has a PhD in computer science from Deakin University, Australia. He's a senior member of IEEE. Contact him at yang@deakin.edu.au.*