Data Sci. Eng. (2016) 1(3):149–160
DOI 10.1007/s41019-016-0014-0

A Cryptographically Enforced Access Control with a Flexible User Revocation on Untrusted Cloud Storage

Jongkil Kim[1] · **Surya Nepal**[1]

Received: 20 July 2016 / Accepted: 28 July 2016 / Published online: 6 September 2016

Abstract Cloud storage services have become ubiquitous. A large number of individuals and organizations are using them to store and share data, taking the benefits of mobility and affordability offered by these services. However, secure management of data in cloud storage services, more specifically supporting multi-party sharing in the context of a collaboration, is a challenging problem. The problem is further exacerbated if the data owner does not have any trust on the cloud storage providers and the data need regular updates from collaborating parties. A number of cryptographically enforced secure cloud storage solutions have been proposed to address this problem. One of the key issues with these solutions is the revocation of access to data for invalid users without moving the data (in the era of big data) and relying on the cloud service providers. In this paper, we introduce a cloud storage system that offers cryptographically enforced security. In contrast to other cryptographically protected cloud storage systems, our system supports a fine-grained access control mechanism and allows flexible revocations of invalid users without moving the data and relying on the cloud service providers. Our system employs an attribute-based encryption technique to support a complex access structure that allows a user to define human readable access policies to the data in the cloud storage. In addition, our system supports a flexible revocation scheme that can revoke invalid users directly by updating the revoked users' list or indirectly by updating an epoch counter. The system administrator can choose one of these options flexibly depending on the needs. Our system also allows authorized users to update the encrypted data, and any users accessing such updated data in future can verify whether the data are modified by authorized users.

Keywords Cloud storage · Access control · Attribute-based encryption · Revocation

1 Introduction

Cloud storage services are widely used by individuals and organizations due to the inherent benefits offered by them; for example, affordability, availability, mobility. Globalization and outsourcing in modern organizations and the increasing adoption of the social web in individual's life have increased the needs of sharing data in a collaborative environment. For example, cloud storage services such as DropBox [3], Google Drive [4] and Microsoft OneDrive [5] have been widely used. In such services, users must rely on the service-level agreement (SLA) to entrust cloud service providers to provide a level of protection to their data. However, SLA-based data protection is vulnerable to different types of threats ranging from legal to ethical. Examples of such threats include data sovereignty, third-party data exploitation, insider attack. This means users cannot trust these cloud service providers for their data. Hence, protecting data on such widely used cloud storage solutions remains a primary concern to their users. If the problem is left unaddressed, cloud storage providers might fail to keep users' data secure and a large amount of private and sensitive data might be revealed with very high consequential financial, reputational and operational impact on both users and cloud storage providers.

One of the most promising solutions to protect the data in cloud storage is encryption. A user encrypts the data

✉ Jongkil Kim
jongkil.Kim@data61.csiro.au

[1] CSIRO ICT Centre, Marsfield, NSW, Australia

before uploading it to the cloud and decrypts the data after it is downloaded. An untrusted cloud storage provider can only see the encrypted data. A number of solutions have been proposed [28, 40, 43] using the above-explained cryptographic method to protect user's data from an untrusted cloud storage. These simple solutions do not support multi-party collaboration between organizations and individuals without a help from a trusted third party. To address this shortcoming, attribute-based encryption (ABE) [14, 34] techniques are often utilized in data storage [8, 29, 39, 41, 44] to support a more fine-grained access control mechanism. ABE is a public key cryptographic system, in which a user encrypts the data using the public keys which are shared with other users so that they can decrypt the encrypted data using their private keys. Particularly, users in ABE do not have unique keys as the traditional public key systems. Instead, they have a key associated with their properties. A user can set a complex access policy based on attributes defined in a system for the encrypted data. Only users with attributes that satisfy the access policy can decrypt the encrypted data. The access policy is easy to express and understand by users since ABE uses a Boolean formula as an access policy. Therefore, we adopted an ABE scheme in our system.

One of the challenging problems in applying ABE to cloud storage systems is managing an up-to-date access control list. For example, a user who was once granted access to the data might not be able to access the data due to various reasons such as change in a security level, leaving an organization or becoming malicious. In such cases, the data owner must be able to revoke access to any encrypted data for invalid users. Two possible revocation methods can be considered. The first is an indirect revocation that revokes users by redistributing keys only to valid users after updating ciphertext. The second is a direct revocation in which the system revokes access to ciphertext for invalid users by updating ciphertext and revocation list without key redistributions. In our system, we employ both methods to support a flexible user revocation.

An *indirect* revocation can be achieved in cloud storage solutions using existing ABE-based techniques (e.g., an epoch counter as proposed in [39]). When a user becomes invalid or there is a need to revoke access for a user, the system first increases the epoch counter in the user's key; then updates the corresponding ciphertext using the increased counter; and finally redistributes keys containing the new epoch counter to all valid users. Since invalid users do not receive a key with the new epoch counter, their access to the corresponding data is automatically revoked. However, this revocation method requires the redistribution of new keys to all users every time when a user becomes invalid. Hence, this may cause inefficiency in the system if the revocation events occur frequently.

A *direct* revocation mechanism can achieve a user revocation without redistributing users' private keys. A system can revoke invalid users only by updating a ciphertext and an associated revocation list. After updating the ciphertext, users in the revocation list cannot decrypt the updated ciphertext while other users not in the revocation list still can decrypt it using their pre-owned keys. Broadcast encryption [9, 11, 12, 19] and revocation schemes [13, 20, 26] can be utilized to support direct revocation in the system. Broadcast encryption keeps an access list instead of a revocation list. For a system requiring a more fine-grained access control, non-monotonic ABE [30, 37, 42] is used. These schemes enable to revoke certain attributes in an access policy using the negation. The disadvantage of these direct revocation mechanisms is that the system becomes inefficient as the time required for encryption/decryption increases proportionally to the number of revoked attributes.

Currently, systems which support a fine-grained access control through ABE provide either direct user revocation [6, 23] or indirect user revocation [21, 38, 39]. However, as we pointed out earlier that both revocation techniques have their own advantages and disadvantages. For example, indirect revocation method does not need to keep a long revocation list, but it needs to redistribute keys when the system revokes an invalid user. Similarly, direct revocation method does not need to redistribute keys to all users, but may need to maintain a long revocation list which makes the system inefficient.

In this paper, we introduce a system that offers a cryptographically enforced access control method on an untrusted cloud storage. When a user uploads a file to a cloud storage service, our system takes over the file and encrypts it with a secret symmetric key (e.g., AES). It then encrypts the secret key separately using an ABE scheme. To maintain the integrity of the encrypted data, our system cryptographically signs both ciphertexts. We call the encrypted file and the encrypted key as *data-block* and *meta-file*, respectively. In our system, only users who satisfy the access policies in meta-file can decrypt the secret key. Hence, the access control policies can be enforced without any help from the cloud storage services, and the data file is cryptographically protected. In addition to encrypted secret key, the meta-file contains features that support a hybrid key revocation mechanism which allows both direct and indirect revocation of users. The use of a particular revocation method at any time is determined by the system based on its own requirements (e.g., efficiency). Our system also supports updates in multi-party collaboration environment through writers access control provided in the meta-file. It allows content updates from authorized users with a write access. Our system works with any third-party cloud storage providers such as DropBox, Google

Drive and Microsoft OneDrive without revealing any confidential information to those cloud storage providers.

The rest of the article is organized as follows. Section 2 describes the security goals and functional features of our system. In Sect. 3, we provide a background of cryptographic primitives used by our system. The detailed working of the system is described in Sects. 5 and 6. In Sect. 7, implementation results of our system are provided. This is followed by a brief summary of related works in Sect. 8. The final section concludes the paper by providing potential future works.

2 System Objectives

2.1 Security Goals

Through our system, we aim to provide an access control including both readers and writers control to the data stored in the cloud. Therefore, our system prevents unauthorized users including the cloud provider to read any plain texts of the files that are stored in the cloud. Authorized users can always access and decrypt the files. Though our system does not prevent the encrypted data in the cloud to be modified or replaced by malicious readers or cloud providers, it can detect such modification or replacement from unauthorized writers. We describe the security goals of our system more formally in terms of confidentiality, integrity and availability (CIA) as follows.

For *confidentiality* of the data, we define the following security functions:

Conf.1 Our system *directly* revokes invalid users without redistribution of keys when they are detected. The revoked users no longer can decrypt the encrypted data.

Conf.2 Our system *indirectly* revokes invalid users by redistributing new keys only to valid users. The revoked users no longer can decrypt the encrypted data

Conf.3 Unauthorized users including a cloud storage provider cannot access the plain text of the encrypted file in the cloud.

For *Integrity* of the data, we define the following security function:

Int.1 Modification or replacement of the encrypted file from unauthorized users is detectable.

For *Availability* of the data, we define the following security functions:

Ava.1 Authorized users can decrypt the encrypted data. The access policy can be fine-grained based on users' attributes.

Ava.2 Authorized users can modify the encrypted data.

2.2 Functional Goals

In addition to the security goals defined above, our system also provides the following distinctive functional features:

Flexible User Revocation Our system uses a hybrid revocation mechanism that supports a fine-grained access control with both direct and indirect user revocations. This is achieved by maintaining both a revocation list and an epoch counter. Our system can revoke users by updating their attributes in the revocation list. If the revocation list becomes too long, the epoch counter is increased and then new keys with increased counter are redistributed to all valid users and the revocation list is reset. The decision on which method to use at a particular time depends on the system; hence, the system has an in-built flexibility to select one of them as required.

Support for Collaboration Our system supports multi-party collaborations by providing an access control for both readers (users who can only read the content in files) and writers (who can read and update the content in files). This is possible in our system as it employs an ABE to support a fine-grained access control mechanism and a cryptographic signature to support the modification of authorized writers. The access policies are associated with users' properties that are used to control readers list and reflect the revocation list. An access policy and a writers list are bundled together with an encrypted data as plain text, but any malicious changes to them are detectable by the system and the confidentiality and the integrity of the data are maintained.

No Third-Party Key Management Server (KMS) Our system does not need a third-party key management server that is normally used to store all decryption keys of the encrypted files. The decryption keys are uploaded in the cloud as a meta-file together with the corresponding data, and they are cryptographically protected. This mitigates the problem of key management, one of the hardest part of cryptography and often the Achilles' heel of an otherwise secure system.

It is worth noting that our system uses an administrator who takes a role of Private Key Generator (PKG) and updates ciphertext and users' private keys when users are revoked. One may think that this is equivalent to a key management server, but the role of the administrator is limited in our system. For example, a user can download the data from a cloud storage and decrypt it without any help from the administrator. In contrast to the systems that

use the key management server, a user does not need to receive any cryptographic key from the administrator every time the data are accessed. In addition, a user can upload a file directly to the cloud and other users can access this file without a confirmation from the administrator if the user explicitly made their collaborators aware of the existence of such file. Furthermore, a user does not need to provide any cryptographic key through the secure channel to the administrator. Therefore, the attack to the administrator such as Denial of Service (DoS) is less effective than other systems that use a centralized authority such as KMS and a proxy server. It is also important to note that the administrator does not have any keys except its own private key; this means the burden of key management is removed from the administrator as it does not need any secure database to store keys.

3 Overview of Cryptographic Primitives

3.1 Overview of ABE

Our system adopted attribute-based encryption [34] to support a fine-grained access policy. ABE system is generally classified into two types based on the location of an access policy employed [14]: key policy attribute-based encryption (KP-ABE) and ciphertext policy attribute-based encryption (CP-ABE). In CP-ABE, an access policy is implemented into the ciphertext and private keys are generated based on a key owner's attributes. Therefore, CP-ABE is a more suitable model for a cloud storage since each user has a key that is created based on their own attributes. The data in the cloud storage are encrypted based on an access policy consisting of those users' attributes.

In our system, we utilize an ABE scheme [42] that supports non-monotonic access structure (NM-CP-ABE). A non-monotonic access structure provides a richer access policy by allowing negation of attributes in the policy. For example, professors can include "NOT Student" in an access policy to revoke student users when they encrypt the data.

Some schemes defined in [6, 23] can also revoke an attribute, but only a special attribute such as an identity. That is, if the system wants to revoke a group of people in the access list, the revocation scheme [6, 23] needs to add all identities of revoked users in a revocation list. However, in NM-CP-ABE, a system can simply negate a common attribute they share. For example, if a system wants to revoke students to access a certain data, a system can simply add "NOT Student" instead of adding all students' identities in the revocation list.

In particular, we utilize the NM-CP-ABE suggested by Yamada et al. [42]. This scheme achieves the negation by division. If a user has a revoked attribute, one of the divisors becomes 0 to prevent the decryption. More specifically, the decryption process needs the following calculation for all negated attributes.

$$\prod_{j\in[1,k]} E(i,j)^{\frac{1}{\mathrm{Att}_i-\mathrm{Att}_j}}$$

where $E(i, j)$ is an intermediate result derived by paring computation of a user's key and a ciphertext and k is the number of attributes the user has. Therefore, if the user has an attribute equals to Att_i, the decryption process fails. The above equation shows that the computation of revocation is $O(nk)$, where n is the number of negated attributes. This clearly demonstrates that the system becomes gradually inefficient as the number of negated (i.e., revoked) attributes increases.

In addition to the negation property in the NM-CP-ABE scheme which enables *direct* revocation, we also use an *epoch counter* for *indirect* revocation. We define an *epoch counter* as a standard attribute and apply it as a mandatory condition to all access policies. In particular, the NM-CP-ABE scheme proposed in [42] supports a large universe of attributes, which is suitable for indirect revocation. Since we define an epoch counter as a standard attribute, the size of the universe is quite important. If the size of the universe of attributes that the system supports is small and has to be fixed at the setup time, the whole system requires to be reset when the epoch counter is reached to the upper bound which is fixed. Since [42] supports the arbitrary size for the epoch counter, this problem does not occur in our system. Therefore, the unbounded nature of these attributes makes the system more practical in real-life applications.

3.2 Overview of Identity-Based Signature

The signature scheme is used to achieve the integrity security goal in our system. Particularly, it is used to control a writer's role in our system. When a user (i.e., uploader) uploads a new data to the cloud storage, the administrator must guarantee that this uploaded file is from the original uploader. This can be verified by the signature of the uploader. After the verification, the administrator's signature replaces the uploader's signature in the meta-file to avoid a further change. However, when the data are modified, the encrypted data are re-signed by the modifier, but the meta-file remains the same. Therefore, a reader of the data can verify at any time that the current version of the data is valid using the signatures on the data-block and meta-file in conjunction with writers list in meta-file.

In order to achieve the functional requirements of the signature scheme described above, we employ an identity-based signature scheme [35]. In an identity-based signature scheme, a signer has a signing key (SK_{ID}) associated with its identity and a verifier has a verification key (VK). Once the data are signed by a signer using its signing key, a verifier can check the integrity of the signed data together with the signer's identity by verifying the signature on the data.

There are a number of identity-based signature schemes [16, 31, 32] in the literature to support our requirements. We use the scheme defined in [31] to implement our system due to its simple structure. In this scheme, a user has a key element D_{id} which is randomization of the hash value of its identity Q_{id} (i.e., $D_{\mathrm{id}} = Q_{\mathrm{id}}^{s}$ where s is a random value). Then, P and P^{s} are given as a public value. Since all users can calculate Q_{id}, they can verify the signature using the public values P and P^{s} via pairing computations.

3.3 Overview of Key Homomorphic Encryption

A key homomorphic encryption [10, 39] is used to provide protection from a cached key, which is explained further in a later section. A key homomorphic encryption facilitates the update on the encrypted data using a new symmetric key without decrypting and re-encrypting it. To use the key homomorphism with our symmetric encryption, we utilize a key homomorphic pseudorandom function F as the symmetric encryption algorithm. Using this function, we encrypt the plain text M using a stream cipher. The plain text M is encrypted as $M \cdot F(K,x)$, where K is a symmetric key and x is a nonce such as an initial vector. Then, the symmetric key K can be updated to K' by calculating $\hat{K} = K' - K$ and $F(\hat{K},x)$. Due to the key homomorphism,

$$M \cdot F(K',x) = M \cdot F(K+\hat{K},x) = M \cdot F(K,x) \cdot F(\hat{K},x).$$

Therefore, the data-block can be updated as a new ciphertext with a new key (K') without the need to decrypt and re-encrypt it.

4 Our Proposed System

4.1 Description of System

As our system aims to provide an access control for the data in an untrusted cloud, the data in the system should be self-defendable from untrusted cloud and malicious users but still be able to share among authorized users in a collaborative environment. Figure 1 shows a simple architecture of our system. Although our system does not require a key management server, we need an administrator who maintains the system as discussed earlier. Since we are using a centralized cryptographic primitive, someone should generate keys for users and distribute them. The administrator takes this role. The administrator also needs to update the ciphertext of meta-file if users are revoked. In our system, the administrator must have a master key to decrypt all data to update the meta-file ciphertext when a user is revoked. A master key can be a private key based on the administrator's identity in NM-CP-ABE. All users must allow this identity to have an access when they encrypt data. However, protecting a single key is easier than maintaining a key management server.

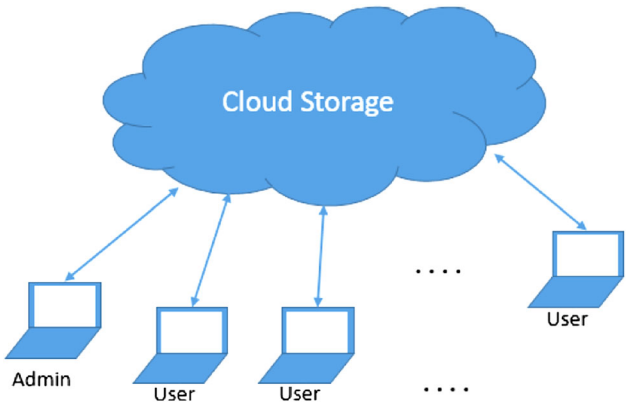

Fig. 1 Architecture of our system

Our system can be implemented as an application that sits between the operating system (e.g., Windows, Linux) and cloud storage application installed in user's device as shown in Fig. 2. In an essence, our application takes over a data stream between a cloud application and an operating system. A cloud application is provided by the cloud storage provider to store/retrieve data to/from the cloud (e.g., an application provided by DropBox). For example, if a user wants to upload a file to the cloud storage, our

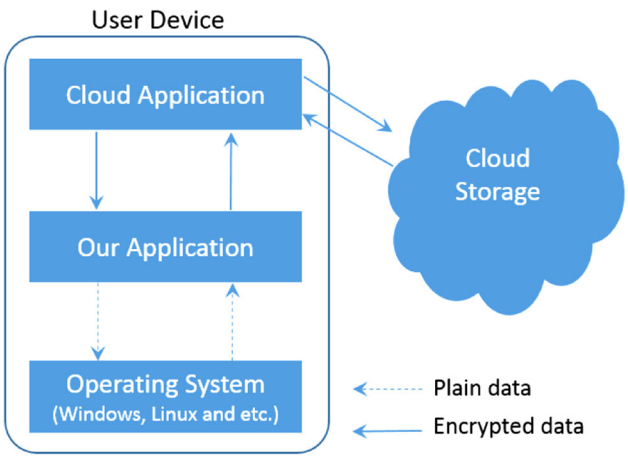

Fig. 2 Implementation in user's device

application takes the file and encrypts it and sends the encrypted file to the cloud application, which in turn stores the encrypted file to the cloud storage. Similarly, when a file is downloaded from the cloud storage, our application receives the encrypted file from a cloud application before it reaches the user. Our application decrypts the file using a user's private key and outputs to the user as a decrypted file.

We utilize an NM-CP-ABE scheme in our system. Each user has attributes including identity (ID) and epoch counter. The system uses these attributes to define an access control policy. Any Boolean formula can be used as an access policy. For example, one can define an access policy as ((“Computer Science” AND “Student”) OR “Professor”) to allow an access to students who study computer science or any professors. In addition, the NM-CP-ABE scheme allows negation of properties. For example, we can use “NOT Student” in an access policy not to allow access particularly for all students. For revocation purpose, we use an *epoch counter*, which is a standard attribute in our system. It works exactly like other attributes, but the epoch counter is mandatory for any access policy. This can be done by binding the value of the epoch counter and an access policy by AND. For example, an access policy with the mandatory epoch counter can be defined as (“1010” AND ((“Computer Science” AND “Student”) OR “Professor”)), where “1010” is the current value of the epoch counter.

In order to support collaboration, our scheme also allows the data modification from authorized users (we refer them as *writers*). This means we need to have a proper protection mechanism in place to prevent unauthorized modifications. As our system operates in the environment where the cloud storage providers are untrusted, the deletion and modification of data in the cloud are unavoidable. However, the system always able to detect unauthorized modifications. In order to achieve this, we use writers' list in the meta-file and protect the integrity of this list using cryptographic signatures from the data owner and administrator. The encrypted data in the data-block is also signed by the one who creates/modifies the data most recently. Using these signatures and writers list, users of data can verify at any time whether the data are modified by an authorized writer. This is done by checking whether a data signer is a valid writer.

4.2 Components of the System

Figure 1 shows different components of our systems such as users, cloud storage and the administrator. We next describe their roles in the overall operation of the system.

4.2.1 Users

Users in our system include authorized devices (owned by individuals) that can download the encrypted data and upload a new or modified data. Examples of devices include laptops, smart phones, PCs. These devices must be powerful enough to compute encryption and decryption under the NM-CP-ABE scheme.

Each user has its own secret key which is created based on their attributes and Identity, and public keys to be used to encrypt and decrypt the data. In addition, each user has its own signing key which is uniquely allocated to its identity and a verification key to verify the signatures signed by other users.

4.2.2 A Cloud Storage

A cloud storage is the remote storage provided by cloud providers that users can upload their data to share with others and also download the data shared by others. We assume that the cloud storage maintains the consistency of the stored data and provides a reasonable service to users. Cloud storage providers may provide some mechanisms to protect data from potential data loss and an unauthorized access to the stored data via SLA. It is important to note that an access control mechanism in our system does not depend on the underlying supports provided by cloud providers. In our system, any information in encrypted data is not revealed to cloud storage providers even when the data are updated and created.

4.2.3 Administrator

Administrator can be one of the users, but it performs a number of maintenance activities for the data stored in the cloud storage as follows:

Key generation and distribution To set up the system, the administrator generates public and private keys using the NM-CP-ABE scheme, and signing and verification keys using the signature scheme discussed earlier. The administrator generates private keys for all users including itself based on their identities and attributes. Then, it distributes the generated keys to corresponding users. The details of key generation and distribution scheme are not covered in this paper. We assume that there exists a secure channel between the administrator and other users for key distribution. That is, keys are delivered securely without leaking them to the cloud service providers or malicious users. The key generation and distribution activity are also performed during the indirect revocation. The administrator increases the

epoch counter for new keys and redistributes them only to valid users.

Approval of a new data When a new data are uploaded into the system, the data contain writers list that is signed by the uploader's identity. An uploader is a user who uploads the data to the cloud storage. However, other users of this data do not know whether the uploader of the data is valid. For example, the uploader could be any reader who can decrypt the file. A reader may generate a new data with a new access policy and writers list and replace the original data. To avoid this unauthorized upload, the meta-file should be re-signed by the administrator, which is trusted by all users, at the beginning. That means the uploader sends a notification to the administrator with a file index when new data are uploaded. Then, the administrator signs these initial data as soon as they are uploaded.

Updating meta-file When a revocation of users is requested, the administrator updates the access to data either by updating the corresponding revocation list or by increasing the corresponding *epoch counter*. In our system, it is not necessary to re-encrypt the entire data. We will explain it further in a later section. However, it should re-encrypts the meta-file with a new policy (i.e., either with a new revocation list or with a new epoch counter). This re-encryption process also requires the decryption of a meta-file since the administrator does not store any individual key for the data-blocks. It is thus necessary for an administrator to have a key which can decrypt all meta-files.

5 System Design

5.1 Data-Block and Meta-File Creation

In this subsection, we aim to provide the detailed description of data-block and meta-file, including how they are created. Figure 3 shows the overall structure of the data-block and the corresponding meta-file. We use a symmetric encryption algorithm (e.g., AES), a NM-CP-ABE scheme [23] and an identity-based signature scheme [32] to realize our system. Before data are uploaded to the cloud storage, each user encrypts the data using the symmetric encryption algorithm with a randomly generated key and signed with the file owner's ID. The symmetric key is not related to user's ID and used only once. We call this encrypted and signed data a *data-block*. Together with a *data-block*, a *meta-file* is created. The meta-file contains the encryption of the symmetric key. It is encrypted using the NM-CP-ABE scheme to support complex access control policies. The meta-file also includes an access policy and writers list as shown in Fig. 3. All those components of meta-file are singed by an uploader's identity.

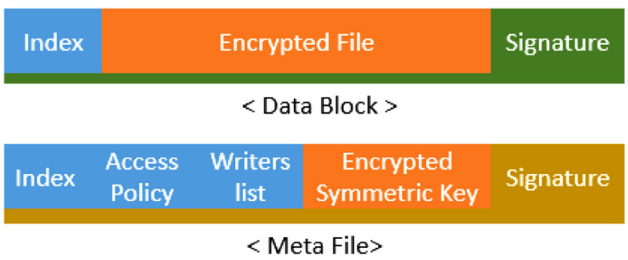

Fig. 3 Diagrams of a data-block and a meta-file

5.2 Assigning Identities and Attributes

In our system, each user has a number of *attributes* including *identity* and *epoch counter*. An *identity* is uniquely assigned to each user. Identity attribute is used for a revocation. When a user becomes invalid, it is a quite difficult sometimes to set up a policy to revoke only a single user since its attributes may be common with other users. In this case, the identity attribute provides a simple way to revoke them. Since the scheme we adopted supports a large universe of attributes, using user's identity as an attribute does not cause a serious scalability problem. In addition, user's identity is also used to sign the data. Hence, associated with identity, each user receives two cryptographic parameters: encryption key element for ABE scheme and signing key for the signature. A user must always sign the data using the assigned signing key after the encrypted data are modified.

5.3 Data Creation

To upload a file into the cloud storage, a user creates a meta-file and a data-block using the following steps:

1. First, a user generates a symmetric key (Key_{sym}) using a random number generator and encrypts the data.
2. After the data are encrypted, the user generates a unique index. The index is then appended to the encrypted data. The user then generates a signature of the index and encrypted data with its signing key (SK_{ID}). The signature is also appended to the data. Therefore, a data-block consisting of an index, an encrypted data and a signature is generated (see Fig. 3).
3. The user uploads the data-block to a cloud storage.
4. The user then sets up an access policy for the data-block. In order to do this, the user checks the revocation list and the current epoch counter in the system. This information can be transmitted from the

administrator to the user in plain text, but with the administrator's signature (to check the integrity of the data received). The user checks whether the revocation list and the epoch counter are valid by verifying the administrator's signature using VK.

5. The user encrypts the symmetric key using the NM-CP-ABE scheme with PK. It should be noted that the output also must contain the access policy for decryption. The user also appends the index which is allocated to the data-block and a writers list to the encrypted symmetric key.
6. The user then generates a signature of the index, the access policy, the writers list and the encrypted key. It is then appended by the signature (see Fig. 3). This is called a meta-file. The user uploads the meta-file and sends the administrator the index of a new file.
7. If the administrator is notified that a new file is uploaded, the administrator downloads the meta-file, verifies the signature of the creator and replaces the signature on the meta-file by its own signature and re-uploads the signed meta-file to the cloud.

In summary, our system uploads a file to the cloud by uploading the corresponding data-block and meta-file. They can be uploaded separately for the efficiency reason, as those processes can be performed in parallel. We also replace the data owner's signature by the administrator's signature. Both signatures are required. The administrator must be able to check whether the meta-file is not replaced by any other users by verifying the uploader's signature before it re-signs the meta-file. Therefore, other users can confirm that the meta-file is the same as the original created by the data owner using the administrator's signature.

5.4 Data Read

To read the data from the cloud storage, a user must decrypt both meta-file and data-block corresponding to the data.

1. A user first downloads both meta-file and data-block corresponding to the file. The meta-file is first downloaded. Therefore, steps 2 and 3 can be performed while the data-block is being downloaded.
2. The user verifies the signature of the meta-file using VK to check the integrity of the meta-file and the signer of the meta-file. If the signature is forged, the user reports it to the administrator.
3. The user then reads the access policy in the meta-file to check whether it is a valid user. If the user is valid, it decrypts the encrypted key using SK_{Att} and PK to get Key_{sym}. Otherwise, it aborts.
4. The user decrypts the encrypted data in the data-block using Key_{sym} retrieved from the meta-file.

5.5 Data Modification

Our system allows a modification of the data by authorized users. Our system does not require the modification of the corresponding meta-file as the symmetric key encrypted in the meta-file remains unchanged while the corresponding data-block is changed.

1. When an authorized user wants to replace or write data, the user reads the data as described in the previous subsection.
2. After the data are modified, the user re-encrypts the modified data with the same symmetric key.
3. The user signs the newly encrypted data using its signing key and then uploads the data to the cloud storage.

During the modification, we need to consider the cached key problem, which will be described further in the next section.

5.6 Revoking Users

Our system revokes users in two different ways as explained earlier: *direct* and *indirect*. When a malicious user is detected or a revocation request is received, the administrator uses either *direct* or *indirect* revocation method to revoke the user. For a *direct revocation*, the administrator revokes users as follows:

1. The administrator downloads all meta-files which are accessible from those users from the cloud and decrypts them to get the symmetric keys for the corresponding data-blocks.
2. The administrator updates the access policies in all meta-files by adding invalid users or attributes and re-encrypts the meta-files.
3. The administrator then updates the revocation list it has.

It should be noted that the above steps are executed in the administrator's device. Hence, the data are still protected from the cloud. Furthermore, all users' keys remain the same. It means redistribution of keys is not necessary in *direct revocation*. For an *indirect revocation*, the administrator revokes users as follows:

1. The administrator generates new keys for all valid users. Users' attributes in those new keys are the same as the previous keys except the epoch counter. All keys share the same epoch counter which is increased by 1.
2. The administrator downloads all meta-files from the cloud storage and decrypts them to get the symmetric keys for the corresponding encrypted data-block.

3. The administrator increases the *epoch counter* by 1, resets the revocation list and re-encrypts them with the updated *epoch counter* and the new revocation list.
4. After the meta-file is updated, it distributes new private secret keys to all users except invalid users.

Since all meta-files are updated with the increased epoch counter, users who do not receive key updates cannot decrypt the meta-file. Similar to the *direct revocation*, the cloud storage providers cannot access the data while executing the above steps.

6 Security Analysis

We perform the security analysis to explain how our system satisfies the confidentiality, integrity and availability properties defined in Sect. 2.

Conf.1 Our system supports the direct revocation using the NM-CP-ABE scheme. Since users in the revocation list are negated in access policies of all meta-files by re-encryption, they no longer can decrypt the meta-files. Hence, they cannot obtain the symmetric keys to decrypt encrypted data-blocks.

Conf.2 Our system supports the indirect revocation using an epoch counter. Since the revoked users do not have a key element associated with an increased epoch counter, they no longer can decrypt meta-files which are re-encrypted using the up-to-date epoch counter. It implies that they do not have the symmetric keys to decrypt the corresponding encrypted data-blocks. Hence, the encrypted data remain confidential.

Conf.3 An unauthorized user and the cloud storage providers do not have a valid key to decrypt any meta-files since they are either negated in the access policy or not have an up-to-date epoch counter as an attribute. Therefore, they cannot get the symmetric key to decrypt any data-block in the cloud. Therefore, the plain text of the data-blocks cannot be revealed to them.

Int.1 When an unauthorized user modifies or replaces the data-block, it is detected by the signature scheme. If an unauthorized user signs with its identity to bypass the integrity check, the readers still can detect it as the writers' list in the meta-file is protected by the administrator's signature.

Ava.1 An authorized user can decrypt the encrypted key in the meta-file. Since the NM-CP-ABE scheme is used to encrypt the key, authorized users can access the keys through a fine-grained access control mechanism supported by the scheme. With the decrypted symmetric key, a user can decrypt the encrypted data in a data-block.

Ava.2 Authorized writers can read the data to modify. They have a key to decrypt the encrypted data in a data-block. Therefore, after they modify the decrypted data, they can encrypt the data with the same key and sign the new data-block by their identity. Since the encryption key of the data-block is the same key which is encrypted in the meta-file and the authorized writer's identity is in the writers' list, the generated data-block is valid.

6.1 A Cached Key Problem

Before the system updates the meta-files to revoke malicious users, the malicious users may cache the symmetric keys to use them after they are revoked. Since the symmetric keys remain the same after the meta-files are updated, they still can access the plain text of the encrypted data even after they are revoked. It is also important to note that the encrypted data are modified by authorized writers without changing the symmetric key. This means they even can read the future modifications of the data using the cached keys. We refer this as a *cached key problem*. To prevent this problem, we update the symmetric key which is encrypted in a meta-file. The change of the symmetric key implies the need to change the encrypted data (i.e., re-encryption). However, decrypting and re-encrypting all data-blocks are a huge burden to the system. To address this problem, we use a key homomorphic algorithm [10] as suggested by Sieve [39] to improve the efficiency of the system. If we use the key homomorphic algorithm, decryption and re-encryption of all data-blocks are not needed. Instead, it only needs to regenerate a fresh randomness to update the encrypted data in a data-block. Moreover, the process does not reveal the plaintext of the encrypted data to the cloud since it does not decrypt the data-block. This update of the symmetric encryption is triggered after the detection of a malicious user. Therefore, the update is performed by the administrator. The administrator is able to distinguish files which were accessible from revoked users and performs the update selectively to those files.

7 Evaluations

We have implemented our system and performs its evaluation. We utilized the Pairing-Based Cryptography (PBC) library [24] and PyCrypto [22] to implement our system. PBC library uses GMP library [1] for mathematical computation. We measured the time to run the NM-CP-ABE scheme [42], identity-based signature scheme [31], SHA256 and AES. It should be noted that our measurements are for individual components. However, using our

results we can approximate the time to generate the meta-file and data-block. For example, to upload a single-file, a user needs to perform one NM-CP-ABE scheme encryption, one AES encryption, two SHA256 hash and two signing.

We implemented our system in Ubuntu 16.04 LTS running on VirtualBox [2]. We only allocated 4-GB memory and 2 processors to run for Ubuntu on the laptop having 16-GB memory and Intel i7 processor. We evaluated the NM-CP-ABE and identity-based signature schemes using PBC with a type A pairing and implemented hash function (SHA256) and symmetric algorithm (AES256-CBC mode) using PyCrypto.

To evaluate the NM-CP-ABE scheme, we gradually increased the number of attributes in keys and ciphertext. Since the NM-CP-ABE scheme supports a large universe of attributes, the execution time of setup algorithm does not depend on the number of attributes. In our experiment, it takes only 16.6 ms. The time to run other algorithms (e.g., key generation and encryption) increases linearly with the number of attributes used in the algorithms. Table 1 shows the time lapsed to generate a key and a ciphertext. More precisely, the number of attributes in encryption means that the number of rows in an access matrix used for the encryption. Decryption process only uses 3 attributes to grant an access, which includes one attribute for access and two attributes for revocation. Although the number of rows used for decryption is fixed, the computation time for decryption increases linearly since the decryption process is also linear to the number of attributes a user has.

We also evaluated the signature scheme of [31], which is adopted in our system. The implementation of this scheme is already given as an example in PBC. We ran this implementation and measured the time of each algorithm consisting of the signature scheme, separately. The results are shown in Table 2.

We additionally implemented SHA256 and AES algorithms using PyCrypto. We measured the running time against the different data sizes. Table 3 shows the results. It is worth noting that the running time of the NM-CP-ABE and signature schemes are not impacted by the size of a file. They are only encrypting AES keys (NM-CP-ABE) or signing hash outputs (signature).

Table 1 Evaluation of NM-CP-ABE [42]

♯ of Att.	KeyGen (ms)	Encryption (ms)	Decryption (ms)
5	61	49.4	53.6
10	101.8	86.6	87
15	151.2	124.6	116.8
20	199.6	153.8	149.4

Table 2 Evaluation of identity-based signature [31]

KeyGen	Sign	Verification
16.8 ms	8.8 ms	7.45 ms

8 Related Works

There are a number of systems [28, 29, 40, 43] that use cryptographic techniques to protect data in a cloud storage. In these systems, the data in the cloud are encrypted, but the access policies are not protected cryptographically. Those systems use a trusted third party such as a key management server or a proxy server to manage the access control. Furthermore, the availability of the system depends on the availability of both the key management sever and cloud data storage.

ABE was often used to support a complicated access policy in the cloud storage services [8, 29, 44]. However, schemes like [8, 44] do not support user revocations. That is, if there is a malicious user detected in the system, they have to reset their encryption system to revoke the user. Systems like [29] allow a revocation, but it requires a third party (i.e., a distributed hash table) to support the revocation.

An attribute-based encryption system is already equipped with a *direct* revocation [6, 17, 23] and an *indirect* revocation [33, 39] methods. In the *direct* revocation mechanism, it uses a revocation list; users in the list cannot decrypt ciphertext as their access is revoked. In all of these schemes, a length of ciphertext and a computation of decryption increase linearly [6, 17, 27] or sublinearly [23] with the number of revoked users. The revocation is applicable only to users' identities. Those identities cannot be used for allowing access in an access policy unless it additionally defines a new identity as a standard attribute.

In an *indirect* revocation system, it uses a counter to revoke users' access to the data. This counter is applied as a mandatory condition to all access policies by setting a policy like "*(counter)* AND *(access policy)*". Sahai et al. [33] additionally shows that the revocation can also be performed by a third party without revealing any secret using a ciphertext delegation and piecewise private keys. However, their scheme still needs key updates to allow valid users to decrypt the updated ciphertext.

Table 3 Evaluation of SHA256 and AES

The size of data	SHA256 (ms)	AES.Enc (ms)	AES.Dec (ms)
45 kbytes	0.3	1.2	1.5
540 kbytes	2.8	10.0	11.1
1.1 mbytes	8.7	19.8	19.2
2.2 mbytes	12.7	23.5	27.6
3.1 mbytes	26.7	34.4	43.3

Predicate encryption (PE) is another approach used for access control. However, a Boolean policy in PE must be converted into a polynomial to support the access control [18]; this puts an additional burden on computations. PE schemes [7, 18, 36] require the fixed maximum degree of polynomial which restricts the size of access policy. GORAM and A-GORAM were introduced in [25] using PE that supports attribute-hiding to provide a stronger privacy. Therefore, a cloud server does not know the access policies that are applied to the data. They also support to read and write the data shared in the cloud like ours, but these systems do not support a user revocation.

Li et al. [21] suggested a solution comparable to our scheme for personal health records. It supports a fine-grained access control using ABE, user revocation and writers' access control. Major difference from our system is that they utilized a multi-authority ABE based on KP-ABE. Therefore, in their system, a ciphertext is encrypted based on a set of attributes and an access policy is in a user's key. This means it does not provide a flexible revocation like ours. Furthermore, their system needs key updates when a user is revoked.

Very recently, the cloud system Seive [39] was introduced. Data in Sieve system are also protected cryptographically. Seive allows a fine-grained access control. However, Seive does not allow a user to modify the data. Furthermore, it only uses indirect revocation, which means it needs to redistribute the key when users are revoked.

AAuth [38] is a variant of OAuth [15]. It applied ABE to OAuth for an access control. The general feature of AAuth is similar to Seive, but it does not provide direct revocation and relies more parties to enforce the access control. For example, when users want to decrypt the file in the cloud, they need to be authorized from an authorizer (on behalf of the data owner). The encrypted file is only accessible after the successful authorization by the server.

9 Conclusion

We introduced a secure cloud storage system that offers a fine-grained access control. Our access control mechanism is cryptographically enforced by ABE. In particular, using ABE with non-monotonic access structure, our system offers a more flexible hybrid revocation scheme which is a combination of both *direct* and *indirect* revocation schemes. In addition, our system allows users to detect an unauthorized modification of data using the signature scheme. We also showed that our system is efficient enough to use in practical applications. We plan to implement the system in existing cloud storage applications like Google Drive, DropBox. In terms of the scheme, our future plan is to further decentralize the role of administrator so that we can update the files in the cloud in an efficient manner.

References

1. https://gmplib.org/. Accessed 07 June 2016
2. https://www.virtualbox.org/. Accessed: 07 June 2016
3. DropBox. http://www.dropbox.com. Accessed: 04 June 2016
4. Google Drive. https://www.google.com.au/drive. Accessed: 04 June 2016
5. OneDrive. https://onedrive.live.com. Accessed: 04 June 2016
6. Attrapadung N, Hideki I (2009) Conjunctive broadcast and attribute-based encryption. In: Shacham H, Waters B (eds) Pairing, volume 5671 of LNCS. Springer, pp 248–265
7. Attrapadung N, Libert B (2012) Functional encryption for public-attribute inner products: achieving constant-size ciphertexts with adaptive security or support for negation. J Math Cryptol 5(2):115–158
8. Baden R, Bender A, Spring N, Bhattacharjee B, Starin D (2009) Persona: an online social network with user-defined privacy. In: Rodriguez P, Biersack EW, Papagiannaki K, Rizzo L (eds) ACM SIGCOMM. ACM, pp 135–146
9. Baek J, Safavi-Naini R, Susilo W (2005) Efficient multi-receiver identity-based encryption and its application to broadcast encryption. In: Public key cryptography. pp 380–397
10. Boneh D, Lewi K, Montgomery HW, Raghunathan A (2013) Key homomorphic prfs and their applications. In: Canetti R, Garay JA (eds) CRYPTO, volume 8042 of LNCS. Springer, pp 410–428
11. Fiat A, Naor M (1993) Broadcast encryption. In: CRYPTO. pp 480–491
12. Gentry C, Waters B (2009) Adaptive security in broadcast encryption systems (with short ciphertexts). In: EUROCRYPT. pp 171–188
13. Goodrich MT, Sun JZ, Tamassia R (2004) Efficient tree-based revocation in groups of low-state devices. In: CRYPTO. pp 511–527
14. Goyal V, Pandey O, Sahai A, Waters B (2006) Attribute-based encryption for fine-grained access control of encrypted data. In: Juels A, Wright RN, De Capitani di Vimercati S (eds) ACM conference on computer and communications security. ACM, pp 89–98
15. Hardt D (2012) The oauth 2.0 authorization framework. RFC 6749
16. Hess F (2002) Efficient identity based signature schemes based on pairings. In: Nyberg K, Heys HM (eds) SAC. pp 310–324
17. Horváth M (2015) Attribute-based encryption optimized for cloud computing. In: Italiano GF, Margaria-Steffen T, Pokorný J, Quisquater J-J, Wattenhofer R (eds) SOFSEM, volume 8939 of LNCS. Springer, pp 566–577
18. Katz J, Sahai A, Waters B (2008) Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: EUROCRYPT. pp 146–162
19. Kim J, Susilo W, Au MH, Seberry J (2013) Efficient semi-static secure broadcast encryption scheme. In: Cao Z, Zhang F (eds) Pairing, volume 8365 of LNCS. Springer, pp 62–76

20. Lewko AB, Sahai A, Waters B (2010) Revocation systems with very small private keys. In: IEEE symposium on security and privacy. pp 273–285
21. Li M, Shucheng Y, Zheng Y, Ren K, Lou W (2013) Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. IEEE Trans. Parallel Distrib. Syst. 24(1):131–143
22. Litzenberger D (2014) PyCrypto: the python cryptography toolkit. https://www.dlitz.net/software/pycrypto/
23. Liu Z, Wong DS (2015) Practical ciphertext-policy attribute-based encryption: traitor tracing, revocation, and large universe. In: Malkin T, Kolesnikov V, Lewko AB, Polychronakis M (eds) ACNS, volume 9092 of LNCS. Springer, pp 127–146
24. Lynn B (2007) On the implementation of pairing-based cryptosystems, Ph.D. thesis, Ph.D. thesis, Stanford University
25. Maffei M, Malavolta G, Reinert M, Schröder D (2015) Privacy and access control for outsourced personal records. In: IEEE symposium on security and privacy. IEEE Computer Society, pp 341–358
26. Naor D, Naor M, Lotspiech J (2001) Revocation and tracing schemes for stateless receivers. In: CRYPTO. pp 41–62
27. Narayan S, Gagné M, Safavi-Naini R (2010) Privacy preserving EHR system using attribute-based infrastructure. In: Perrig A, Sion R (eds) ACM CCSW. ACM, pp 47–52
28. Nepal S, Sinnott R, Friedrich C, Wise C, Chen S, Kanwal S, Yao J, Lonie A (2015) Truxy: trusted storage cloud for scientific workflows. IEEE Trans Cloud Comput PP(99):1
29. Nilizadeh S, Jahid S, Mittal P, Borisov N, Kapadia A (2012) Cachet: a decentralized architecture for privacy preserving social networking with caching. In: Barakat C, Teixeira R, Ramakrishnan KK, Thiran P (eds) CoNEXT. ACM, pp 337–348
30. Ostrovsky R, Sahai A, Waters B (2007) Attribute-based encryption with non-monotonic access structures. In: Ning P, De Capitani di Vimercati S, Syverson PF (eds) ACM conference on computer and communications security. ACM, pp 195–203
31. Paterson KG (2002) Id-based signatures from pairings on elliptic curves. IACR Cryptol ePrint Arch 2002:4
32. Paterson KG, Schuldt JCN (2006) Efficient identity-based signatures secure in the standard model. In: Batten LM, Safavi-Naini R (eds) ACISP, volume 4058 of LNCS. Springer, pp 207–222
33. Sahai A, Seyalioglu H, Waters B (2012) Dynamic credentials and ciphertext delegation for attribute-based encryption. In: Safavi-Naini R, Canetti R (eds) Advances in cryptology—CRYPTO 2012—32nd annual cryptology conference, Santa Barbara, CA, USA, August 19–23, 2012. Proceedings, volume 7417 of LNCS. Springer, pp 199–217
34. Sahai A, Waters B (2005) Fuzzy identity-based encryption. In: EUROCRYPT. pp 457–473
35. Shamir A (1984) Identity-based cryptosystems and signature schemes. In: CRYPTO. pp 47–53
36. Shen E, Shi E, Waters B (2009) Predicate privacy in encryption systems. In: Reingold O (ed) TCC, volume 5444. Springer, pp 457–473
37. Takashima K (2014) Expressive attribute-based encryption with constant-size ciphertexts from the decisional linear assumption. In: Abdalla M, De Prisco R (eds) SCN, volume 8642 of LNCS. Springer, pp 298–317
38. Tassanaviboon A, Gong G (2011) Oauth and abe based authorization in semi-trusted cloud computing: aauth. In: Proceedings of the second international workshop on data intensive computing in the clouds, DataCloud-SC '11. ACM, pp 41–50
39. Wang F, Mickens J, Zeldovich N, Vaikuntanathan V (2016) Sieve: cryptographically enforced access control for user data in untrusted clouds. In: NSDI, Santa Clara, CA, March 2016. USENIX Association. pp 611–626
40. Wise C, Friedrich C, Nepal S, Chen S, Sinnott RO (2015) Cloud docs: secure scalable document sharing on public clouds. In: Pu C, Mohindra A (eds) IEEE CLOUD. IEEE, pp 532–539
41. Xu Z, Martin KM (2012) Dynamic user revocation and key refreshing for attribute-based encryption in cloud storage. In: Min G, Wu Y, Liu L, Jin X, Jarvis SA, Al-Dubai AY (eds) TrustCom. IEEE Computer Society, pp 844–849
42. Yamada S, Attrapadung N, Hanaoka G, Kunihiro N (2014) A framework and compact constructions for non-monotonic attribute-based encryption. In: Krawczyk H (ed), PKC, volume 8383 of LNCS. Springer, pp 275–292
43. Yao J, Chen S, Nepal S, Levy D, Zic J (2010) Truststore: making amazon S3 trustworthy with services composition. In: IEEE/ACM CCGrid. IEEE Computer Society, pp 600–605
44. Zhang L, Mislove A (2013) Building confederated web-based services with priv.io. In: Muthu MS, Abbadi AE, Krishnamurthy B (eds) COSN. ACM, pp 189–200