Towards Efficient Update of Access Control Policy for Cryptographic Cloud Storage

Weiyu Jiang[1,2,3(✉)], Zhan Wang[1,2], Limin Liu[1,2], and Neng Gao[1,2]

[1] State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{wyjiang,zwang,lmliu,gaoneng}@lois.cn
[2] Data Assurance and Communication Security Research Center
of Chinese Academy of Sciences, Beijing, China
[3] University of Chinese Academy of Sciences, Beijing, China

Abstract. To protect sensitive data from unauthorized access, encrypting data at the user end before outsourcing them to the cloud storage, has become a common practice. In this case, the access control policy is enforced through assigning proper cryptographic keys among collaborators. However, when the access control policy needs to be updated (e.g. new collaborators join or some collaborators leave), it is very costly for the data owner or other parties to re-encrypt the data with a new key in order to satisfy the new policy. To address this problem, we propose a dual-header structure and batch revocation, which makes the overhead for privileges grant independent of data size and significantly improves the efficiency of privilege revocation by applying lazy revocation to certain groups of revocation requests, respectively. We also analyze the overhead for authorization showing that our approach is able to efficiently manage frequent policy updates.

Keywords: Access control policy · Over-encryption · Batch revocation

1 Introduction

Cloud storage has certain advantages, such as paying for only what is used, being quick to deploy, offering easy adjustment of capacity and built-in disaster recovery. Therefore, individuals and companies are resorting more to cloud providers for storing their data and sharing them with collaborators. However, cloud providers are generally considered as "Honest-but-Curious", which means that the cloud will execute some functions honestly, but might pry into the sensitive data led by business interest or curiosity. To secure sensitive data and prevent illegal visitors (including cloud providers) from unauthorized access, a straightforward solution is to apply cryptographic techniques, so that data are encrypted at the user end before being outsourced to the cloud. In this case, only the data owner and authorized collaborators with knowledge of the key will be able to access the data. Therefore, access control policies are enforced through assigning proper cryptographic keys among collaborators.

J. Tian et al. (Eds.): SecureComm 2014, Part II, LNICST 153, pp. 341–356, 2015.
DOI: 10.1007/978-3-319-23802-9_26

However, when the access control policy needs to be updated (e.g. new collaborators join or some collaborators leave), it can be very costly for data owners to re-encrypt the data with a new key in order to satisfy the new policy. As the computation overhead for re-encryption(encryption/decryption) and transmission overhead for downloading are proportional to the size of data [1], policy updates may not propagate in real time, especially for large amounts of data. Therefore, it is not advisable for data owners with limited ability to take the heavy burden. An alternative solution is applying proxy re-encryption [2,3] which migrates the burden for re-encryption from data owners to the proxy. However, the adoption of public key cryptography impedes the wide usage of proxy re-encryption algorithms, because of the computation overhead. Over-encryption proposed in [1,4] is a practical symmetric encryption solution for delegating keys' update and re-encryption to cloud servers. Nevertheless, in the "pay-as-you-go" model of cloud computing, it is still costly for data owners to pay the cloud for the cipher operations. Furthermore, the delay for re-encryption cannot be ignored, especially in presence of multiple access control policy updates of large data with replicas across multiple cloud servers.

Our approach is based on over-encryption [1,4], which implements the update of access control policy by enforcing two layer encryption. In over-encryption, data resources are doubly encrypted at base encryption layer (BEL) by data owners and at the surface encryption layer (SEL) by the cloud. When access control updates, the data just needs to invoke the cloud to update the encryption policy at SEL. However, both granting and revoking authorizations need the cloud to encrypt over the pre-encrypted data, which brings much overhead for re-encryption computation and has an influence on the performance when large amounts of updating operations of access control policy concurrently happen. In order to implement an efficient update of access control policy in cryptographic cloud storage, this paper presents a dual-header structure for eliminating the need of re-encrypting related data resources when new authorizations are granted, and proposes batch revocation for reducing the overhead for re-encryption when revocations happen.

In our dual-header structure, data are encrypted by data owners at the base encryption layer and then over-encrypted by cloud servers at the surface encryption layer. Each data resource is divided into the data content in the body and the cryptographic keys of data content in the header. Before being outsourced to the cloud, both the body and the header of data resources are pre-encrypted by data owners. After data are uploaded to the cloud, the cloud server will first encapsulate the header by encryption. Therefore, the header of all the resources is initialized by a two layer encryption and always has a relatively small size. When granting new privileges, cloud servers only need to update the small header, instead of the body. Our dual-header structure has the following characteristics:

– High security. The dual-header structure prevents unauthorized visitors from accessing the sensitive data. Even if the cloud server suffers attacks, the sensitive data will not be divulged to unauthorized visitors.

– Low overhead. The dual-header structure makes the overhead for granting privileges independent of data size. With the dual-header structure, there is no re-encryption of any data content (possibly of large size), so it offers significant benefits in reducing the overhead when new privileges are granted.

In order to prevent the revoked user from accessing future versions of the data with the key they possess, the overhead for re-encryption brought by revocation operations cannot be avoided. Our batch revocation mechanism, combining lazy revocation to a certain group of revocation requests, provides a considerable improvement of over-encryption systems, by reducing the number of operations on large amounts of data.

The rest of the paper is organized as follows. Section 2 introduces a basic scheme and discusses its weaknesses. Section 3 presents our main scheme, and Sect. 4 describes an efficient access control policy update with low overhead. Section 5 illustrates performance analysis and security analysis. Then related work is given in Sect. 6, and finally we conclude this paper and give our future work in Sect. 7.

2 Preliminaries

Cryptographic cloud storage [5] is proposed to securely outsource sensitive data resource to the "Honest-but-Curious" cloud. It can protect sensitive data against both the cloud provider and illegal visitors, by encrypting data at the client side before outsourcing. The security lies in appropriate key distribution to users (collaborators) based on the access control policy for sharing data among collaborators. Keeping cryptographic keys secret from the cloud provider is essential for those data owners with high security requirement. However, it makes it difficult for data owners to resort to the cloud provider for updating the access control policy when the cooperative relationship changes. Additionally, data with different access control policies should be encrypted with different keys when fine-grained data access control is desired. This could upset the users, as they would be required to maintain multiple keys for different data resources.

Our work is based on the over-encryption approach [1,4], which was proposed to avoid the need for shipping resources back to the owner for re-encryption after a change in the access control policy. On the premise of implementing fine-grained access control, over-encryption also forces a user to keep one or two private keys to access all the authorized resources, by subtly constructing a key derivation structure. In over-encryption, data resources are doubly encrypted at the base encryption layer (BEL) and the surface encryption layer (SEL). At BEL, data are encrypted by data owners at client side and data owners are responsible for distributing the decryption keys to users. After data are outsourced to the cloud, the encrypted data are over-encrypted by the cloud at SEL, for updating access control policies. Only those with keys of the two encryption layers can decrypt the data, so the cloud provider offers additional protection to prevent those who can obtain the keys of the base encryption layer from accessing the data.

When the cooperative relationship or the access control requirements of data owners change, the access control policy should be updated as well. In over-encryption, the data owner only needs to call the cloud servers to re-encrypt the data at the surface encryption layer. However, re-encrypting large amounts of data and transmitting requests across multiple servers with replicas are also costly for the cloud when multiple access control policy updates happen. One potential limitation of over-encryption is that the cloud might need to re-encrypt the content of related data resources when new privileges are granted. Another improvable point is that immediate revocation could increase the overhead for repetitive cipher operations, when revoking privileges towards the same resources frequently happens.

2.1 Over-Encryption

In over-encryption, if a set of data resources can be accessed by the same access user set, they will be encrypted with the same key at the base encryption layer, or else they will be encrypted with different keys. A user just needs to maintain one or two private keys to access all the resources that are authorized to him. Over-encryption is implemented by constructing a key derivation structure, where one key can be derived from another key through public tokens.

The key derivation structure of over-encryption is based on the access control list (ACL) of data resources. In which over-encryption divides all the users into different access user sets and each access user set is associated with a key. Data resources with the same access user set are encrypted with the same key. The associated key of the access user set, can be derived by the associated key of any subset of the access user set. It is implemented by publishing public tokens and labels on each derivation path. For example, there are three data resources r_1, r_2 and r_3: the access user set of r_1 is {A,B} with associated key K_{AB}; r_2 and r_3 with the same access user set {A,B,C} are encrypted with the key K_{ABC}; by publishing token $t_{AB,ABC} = K_{ABC} \oplus h_a(K_{AB}, l_{ABC})$, the user who possesses K_{AB} can derive K_{ABC} by computing $K_{ABC} = t_{AB,ABC} \oplus h_a(K_{AB}, l_{ABC})$, where l_{ABC} is a publicly available label associated with K_{ABC}, $\oplus$ is the bita-bit xor operator and h_a is a secure hash function.

We express the key derivation structure through a graph, having the vertex v_U associated with a group of resources and keys to encrypt the resources. If U_i is a subset of U_j and a token $t_{i,j}$ is published, then there exists an edge connecting two vertices (v_{U_i}, v_{U_j}). For instance, Table 1 represents an example of access control policy, where h_d and h_a is a secure hash function, then a key derivation structure shown in Fig. 1 is constructed.

2.2 Limitations

In the key derivation structure, data resources with the same access user set are encrypted with the same key in a vertex. It reduces the number of keys and significantly simplifies key management for users. However, it might result in re-encrypting the other data resources in the same vertex of the granted data

Table 1. An example of the implementation of access control policy

(a) Secret Keys

Resources	Access User Sets	Encryption Keys
r_1, r_9, r_{10}	A,B	$h_d(K_{AB})$
r_3, r_4, r_5	A,B,C	$h_d(K_{ABC})$
r_2, r_6	C	$h_d(K_C)$
r_7, r_8	D	$h_d(K_D)$

(b) Public Tokens

Labels	Tokens
l_{AB}	$t_{A,AB} = K_{AB} \oplus h_a(K_A, l_{AB})$
l_{AB}	$t_{B,AB} = K_{AB} \oplus h_a(K_B, l_{AB})$
l_{ABC}	$t_{AB,ABC} = K_{ABC} \oplus h_a(K_{AB}, l_{ABC})$
l_{ABC}	$t_{C,ABC} = K_{ABC} \oplus h_a(K_C, l_{ABC})$

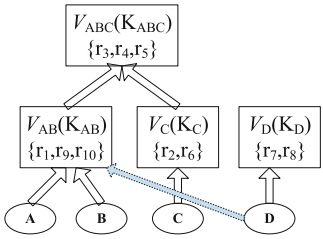

Fig. 1. Key derivation structure

resource when new privileges are granted. In the example showed by Fig. 1, if the data owner grants user D the privilege of accessing the data resource r_1, the data owner needs to provide D with the decryption key $h_d(K_{AB})$ instead of the derivation key K_{AB}, which might be used to derive the key of resources (e.g. r_3, r_4, r_5) in other vertices. However, it cannot prevent unauthorized D from decrypting r_9 and r_{10}. Therefore, the cloud provider should over-encrypt r_9 and r_{10} at the surface encryption layer instead of shipping them back to the data owner. In fact, re-encrypting data resources in the same vertex when granting privileges should be avoided.

Another improvable point of over-encryption lies in revocation. In order to prevent the revoked users from accessing future versions of the data resource with the key they possess, the cloud should re-encrypt it at the surface layer encryption. However, the costly re-encryption operations might affect the performance of the cloud storage service when multiple revocations happen. Moreover, as a data resource might be accessed by a set of users, immediately revoking the access to a certain resource will produce repetitive re-encryption operations, and may result in a long delay when revoking the privileges on large data.

3 Main Scheme

We construct a dual-header structure based on over-encryption and propose batch revocation to implement an efficient update of the access control policy in cryptographic cloud storage. In order to implement fast encryption, we adopt symmetric ciphers in our proposed scheme. Data are firstly encrypted at base encryption layer by data owners. When the access control policy changes, data owners will not re-encrypt the encrypted data any more. All of the cipher operations for matching the new access control policy are executed by the cloud. The

dual-header structure makes the overhead for granting privileges independent of data size. Therefore, the cloud just needs to update a small header of the granted resource, instead of the large content of other resources encrypted with the same key of the granted resource.

3.1 Dual-Header Structure

We divide each data resource into two parts: keys in the header and the data content in the body. At the initialization phase (before uploading), the data content in the body is encrypted with the key in the header by the data owner at the base encryption layer. In our scheme, each resource uses a different key to encrypt its content. In order to prevent the cloud provider and unauthorized visitors from obtaining the secret key, the key in the header at the base encryption layer is encrypted by the data owner. When data resources in header/body form are uploaded to the cloud servers, the cloud needs to over-encrypt the header at the surface encryption layer. Therefore, the two layer encryption is imposed on the header of all the resources and we call it dual-header structure.

There are four types of keys in our dual-header structure.

- Data Content Key: *dek*. This is a symmetric key used in the base encryption layer to encrypt the data content in the body. It is generated and encrypted by the data owner and stays invariant in the header in the cloud. Each data resource has a different data content key. This key is stored in the header in encrypted form and requires no distribution.
- Surface Content Key: *sek*. This is a symmetric key used in the surface encryption layer to encrypt the already encrypted data content in the body. At the initialization phase, it is null. When the revocation of the data resource happens, the cloud will set a new surface content key and encrypt the pre-encrypted data content with it in the body. The keys of separate data resources are also different and will be changed when revocations happen. This key is stored in the header in encrypted form and requires no distribution.
- Base Head Key: BK_U. This symmetric key is used to encrypt the data content key in the header. The data owner also generates it before uploading the header to the cloud and it will also stay invariant in the cloud. It might be used to encrypt a set of resources with the same access control policy. This key is distributed to all the authorized users of set U, by constructing derivation paths from their private keys to BK_U.
- Surface Head Key: SK_U. This symmetric key is used in the surface encryption layer to encrypt the pre-encrypted data content key and surface content key in the header. The cloud generates it and it will change when the access control policy updates. Data resources with the same access control policy share the same surface head key. This key is also distributed to the authorized users of set U, by constructing derivation paths from their private keys to SK_U.

We use the four types of symmetric keys at the two encryption layer to protect the outsourced data. As the access control policy might update, the

status of the data stored in the cloud is not immutable. After the data resource is uploaded to the cloud at the initialization phase, the data resource is in the initial status expressed in Table 2. When the access control policy of the data resource updates, the status of the data will change into the common status showed in Table 3.

Table 2. Initial status of data resource

Id	Header	Body
Id(r)	$E_{SK_{U_i}}(E_{BK_{U_i}}(dek), null)$	$E_{dek}(data)$

Table 3. Common status of data resource

Id	Header	Body
Id(r)	$E_{SK_{U_j}}(E_{BK_{U_i}}(dek), sek)$	$E_{sek}(E_{dek}(data))$

At the initialization phase, the data owner first encrypts the data resource with data content key dek and generates the body $E_{dek}(data)$, then encrypts dek with the base head key BK_{U_i} and achieves the header $E_{BK_{U_i}}(dek)$, and finally uploads $Id(r)$ (the identifier of the data resource r), $E_{dek}(data)$ and $E_{BK_{U_i}}(dek)$ to the cloud. After the cloud receives the data, the cloud first encrypts $E_{BK_{U_i}}(dek)$ in the header with the surface head key SK_{U_i}, and gets $E_{SK_{U_i}}(E_{BK_{U_i}}(dek), null)$ ($null$ means that the cloud has not over-encrypted $E_{dek}(data)$). Then the data resource is stored in the initial status.

When the access control policy changes, the data owner should prevent the users who own dek from accessing the data. If data owners are unwilling to download the data resource and re-encrypt it by themselves, they can invoke the cloud to over-encrypt it. If the data resource is still in the initial status, the cloud needs to generate a surface content key sek and a new surface head key SK_{U_j}, then over-encrypt $E_{dek}(data)$ with sek and re-encrypt $(E_{BK_{U_i}}(dek), null)$ with the new SK_{U_j}. Then the status of the data will change into the common status. If the data resource is in the common status, the cloud will decrypt $E_{sek}(E_{dek}(data))$ with the old sek and re-encrypt it with a new sek.

Our work assumes that each data resource has an access control list ACL. In order to enforce fine-grained access control through reasonably assigning keys, we define the key derivation function $KeyDerivation(U)$ to generate encryption keys, distribute keys to shared users and publish tokens to derive keys for authorized users. For the detailed algorithm code of $KeyDerivation(U)$ refers to [4].

Definition $KeyDerivation(U) \longrightarrow (K, T, L)$:

– Access User Sets U: U is the family of subsets of all the users which derives from the access control lists of all the data resources. For instance, if the access control list of data resource r_i regulates that users $\{A, B, C\}$ can read it, then $U_i = \{A, B, C\}(U_i \in U)$ is the access user set of r_i.

- Keys K: K can be the set of all the keys used to derive the keys of the header (base head key BK_{U_i} or surface head key SK_{U_i}). At the base encryption layer at the initialization phase, $\forall U_i \in U, \exists K_{U_i}$ associated with the access user set U_i, where $BK_{U_i} = h_d(K_{U_i})$. At the surface encryption layer, $\forall U_i \in U, \exists SK_{U_i}$ associated with the access user set U_i.
- Public tokens T and labels L: T is the set of all the public tokens which are used to derive keys for the users. L is the set of all the labels which are used to mark access user sets. If $\exists U_j \in U$ and U_i is the largest subset of U_j among U, then it must exist a token $t_{U_i,U_j} = K_{U_j} \oplus h_a(K_{U_i}, l_{U_j})$ (l_{U_j}) is the label of access user set U_j.

3.2 Batch Revocation

There are two revocation approaches in cryptographic cloud storage, depending on when the re-encryption operations are executed. In an active revocation approach, the revoked data resource is immediately re-encrypted with a new key after a revocation takes place. This is costly and might cause disruptions in the normal operation of cloud storage. In the alternative approach of lazy revocation [6], re-encryption happens only when the data resource is modified for the first time after a revocation.

We propose batch revocation combining lazy revocation to achieve better user experience and reduce the overhead for revocation. In the general scheme, when data owners need to prevent revoked users from accessing their resources, they can invoke the cloud provider to re-encrypt data after a revocation. In this case, revocation operations must involve reading data from the disk, decrypting them and re-encrypting them, so the overhead for revocation cannot be ignored, especially for the data of large size. In our scheme, the cloud can delay the revocations to the time when the predefined conditions are satisfied. The predefined conditions and the final time of revocation can be set by data owners according to their requirements. For example, the cloud can select to delay the revocations on the data of large size to the next read access, which are not frequently accessed. As the base head key is not updated when the data resource is modified, the data owner will use a new data content key to encrypt the content when the data owner modifies it, and the cloud just needs to re-encrypt the header without encrypting the content in the body (the data resource is stored in the initial status). In this case, the cloud can delay the revocations to the next write access in the scenario where multiple revocation operations frequently happen.

4 Access Control Policy Updates

There are two types of access control policy update operations in most storage systems: (1) Grant new privileges to users and (2) Revoke privileges. The privileges can be referred to as read privilege or write privilege. Our target is to protect the sensitive data from being disclosed to unauthorized visitors, and we restrict ourselves to the consideration of read privileges in this paper.

Policy update operations are often executed in most network applications or systems. For instance, according to the data given in [7], there are 29,203 individual revocations of users from 2,916 different access control lists extracted from seven months of AFS protection server logs. If the updating of access control policies requires heavy overhead, it will have a negative influence on the performance. In over-encryption, both granting and revoking involve reading data from the disk, encrypting data resource and decrypting data resource, so it results in large transmission overhead and computation overhead. Our dual-header structure can efficiently reduce the overhead when new privileges are granted, by operating on the small header of the granted resource, instead of the data content with large size. As for revocation, our scheme applies batch revocation to reduce the overhead for repetitive re-encryption operations.

4.1 Granting Privileges

We define the function $Grant(u, r)$ to authorize a user u to access the data resource r in cryptographic cloud storage systems. Privileges grant in our scheme is implemented by assigning the related keys to the authorized users. In the previous work of over-encryption, grant in both Full_Sel and Delta_Sel [4] methods involves encryption and decryption operations on the data resource content and other related resources encrypted with the same keys of r. However, we require no re-encryption of the content and just require the cloud to re-encrypt the header of r.

When executing $Grant(u, r)$, the data owner firstly updates the access user set $r.USet$ of r, then gets the derivation key K according to $r.USet$ and computes the base head key $r.BK$ of r by hashing K. As resources with the same privileges at the initialization phase are encrypted with the same base head key, which is not changed with the access control policy, $r.BK$ may be derived from the private key K_u of u . If the base head key of r is not included in the set of keys $KSet$ which can be derived by u, the data owner has to add token from K_u to $r.BK$, in order to ensure that u can derive $r.BK$. Then the data owner invokes the cloud to over encrypt the header of r to make sure that only the new access user set $r.USet$ can decrypt the header of r. When the cloud receives the request, the cloud needs to decrypt the header of r with the old surface head key, re-encrypt the header and add tokens to ensure that all the authorized users in the access user set of U can decrypt the header at the surface encryption layer, which is implemented by calling the function $ReEncryptHeader(header, U)$. The detailed steps can be seen in Fig. 2.

For the sake of simplicity, we assume that the function $Grant(u, r)$ is referred to a single user u and a single resource r. The extension to sets of users and resources is easy to implement. The main overhead of $Grant(u, r)$ lies in decrypting and re-encrypting the small header of r: $DecryptHeader(r, r.SK)$ and $ReEncryptHeader(r.Header, U_{new})$ in Fig. 2.

Granting new privileges	Revoking privileges
Data Owner:$Grant(u, r)$ 1.$r.USet \leftarrow r.USet \bigcup \{u\}$ 2. $K_{U_i} \leftarrow GetKey(r)$ 3. $r.BK \leftarrow H_d(K_{U_i})$ 4. $KSet \leftarrow FindAllKey(K_u)$ 5. If $r.BK \notin KSet$ Then $AddToken(K_u, r.BK)$ 6. $OverEncryptHeader(r, r.USet)$ **Cloud:**$OverEncryptHeader(r, U_{new})$ 1. If $r.USet \neq U_{new}$ Then $r.SK \leftarrow GetKey(r)$ $r.Header \leftarrow DecryptHeader(r, r.SK)$ $ReEncryptHeader(r.Header, U_{new})$	**Data Owner:**$Revoke(U, r)$ 1.$r.USet \leftarrow r.USet - U$ 2. $OverEncryptResource(r, r.USet)$ **Cloud:**$BatchRevoke(r, U_{new})$ 1. $r.USet \leftarrow U_{new}$ 2. If $T_{curr} \geqslant RevocationTime(r)$ or the predefined conditions are satisfied Then $r.SK \leftarrow GetKey(r)$ $r.Header \leftarrow DecryptHeader(r.Header, r.SK)$ If $r.Header.sek \neq null$ Then $sek_{old} \leftarrow r.Header.sek$ $r.Body \leftarrow DecryptBody(r.Body, sek_{old})$ $r.Header.sek \leftarrow GenNewKey()$ $EncryptBody(r.Body, r.Header.sek)$ $ReEncryptHeader(r.Header, U_{new})$ 3.Else $Wait...$

Cloud:$ReEncryptHeader(header, U)$
1. If $\exists U_i$ is an access user set and $U_i = U$
Then $SK \leftarrow GetSurfaceHeadKey(U_i)$
2. Else $SK \leftarrow GenNewKey()$
3. $EncryptHeader(header, SK)$
4. While $U \neq null$ % Ensure all the users in U can decrypt the header
$U_{max} \leftarrow MaxSubUset(U)$ % Find the maximal access user set U_{max}, $U_{max} \subseteqq U$
$SK_{U_{max}} \leftarrow GetSurfaceHeadKey(U_{max})$
$AddToken(SK_{U_{max}}, SK)$
$U \leftarrow U - U_{max}$

Fig. 2. Algorithms for granting and revoking authorizations

4.2 Revoking Privileges

Revocation in our scheme is implemented by updating the keys and re-encrypting the resource at the surface encryption layer. Users whose privileges will be revoked, might preserve the keys of the related resources locally, therefore the revoked resource should be re-encrypted with new keys. As the cloud could not change the base layer encryption data, we need the cloud to re-encrypt the resource at the surface encryption layer.

We define the function $Revoke(r, U)$ at the client side to revoke a set of users $U(|U| >= 1)$ the access to a resource r. At the cloud side, we define the function $BatchRevoke(r, U_{new})$ to revoke a set of users not in U_{new} on r. When executing $Revoke(r, U)$, the data owner updates the access user set $r.USet$ of r by deleting the revoked users U from $r.USet$, invokes the cloud to over-encrypt r and requires the cloud to ensure that only users in $r.USet$ can access the new decryption keys by executing $OverEncryptResource(r, r.USet)$. When receives the request, the cloud will record the freshest access user set of r and wait for the revocation time to execute the function $BatchRevoke(r, U)$. The data owner can define a time period for resources to execute revocations, then the cloud must execute revocations when the final time arrives. The data owner can also predefine conditions to require the cloud execute re-encryption. When the cloud needs to execute re-encryption for revoked resource r, it has to decrypt the header of r and extract the surface content key sek_{old} of r. If sek_{old} is $null$, it means that the body of r has not been over-encrypted by the cloud. Or the cloud should decrypt the body of r with sek_{old}. Finally, the cloud should encrypt the

body of r with a new surface content key and re-encrypt the header to ensure only the authorized users in the access user set U_{new} can decrypt the header of r. The details are given in Fig. 2.

The main overhead for revocations lies in the two functions *DecryptBody* and *EncryptBody*. For the data resources of large size, the overhead cannot be ignored. However, batch revocation can reduce the number of cipher operations on the resource, which will be illustrated in Sect. 5.1.

5 Analysis

5.1 Performance Analysis

In cryptographic cloud storage systems, the keys to encrypt data resources need to be updated and re-encryption might be required in order to match the new access control policy. However, the overhead for re-encryption could not be ignored, especially for large amounts of data resources in the cloud. For example, encrypting data with the size of 1 GB will consume 7.15 s by applying OpenSSL 0.9.8 k with a block size of 8 KB (AES-256, encoding rate: 143.30 MB/s) [8]. Therefore, our scheme targets at reducing the overhead for re-encryption after the access control policy changes.

The overhead for privileges grant. The overhead for privileges grant in our dual-header structure, always involves token retrieval and key derivation, reading data from the disk and encryption/decryption. At the client side, the dominant computation overhead is the retrieval of tokens and key derivation to distribute keys to the new authorized users when new privileges are granted. At the cloud side, the cloud servers have to find the key of related resources by retrieving tokens and deriving keys, read the related resources from the disk and re-encrypt them.

According to the performance evaluations of over-encryption in the extension work [1], the time for retrieving tokens, independent of resource size, is much lower than that for downloading and decrypting large data resources. However, the time required to transfer and decrypt the resource in [1], dominates in the overhead for authorization on resources of size larger than 1 MB in its local network configuration. The time also grows linearly with the increase in the resource size. Although the cloud does not transfer data resources back to the client, the cloud is required to read the resource from the disk, re-encrypt it and sometimes might transfer re-encryption request among different cloud servers with replicas. Therefore, reading data from the disk, decrypting and encrypting data dominate in the overhead for access control policy updates. As the time for reading data from the disk, decrypting and encrypting data is proportional to the size of data resources, our approach that operating on small (about KB level) header rather than operating on data content (perhaps MB/GB/TB level), has significant benefits in reducing the overhead.

The overhead for revocation. We find that the number of operations of cloud servers on data resources is different between revoking a group of users

Table 4. Comparison of the number of operations on data resource content

Example–Access policy updates :{A,B,C,D,E,F} can read r $\longrightarrow$ $\{A, E, F\}$ can read r Re-encrypt the header or the body of r with a new surface key: K_U, U is the access user set of r		
Function		Main operations
Revoking one by one	$Revoke(B, r)$	$Read(r)$
		$Decrypt(r, K_{ABCDEF})$
		$Encrypt(r, K_{ACDEF})$
	$Revoke(C, r)$	$Read(r)$
		$Decrypt(r, K_{ACDEF})$
		$Encrypt(r, K_{ADEF})$
	$Revoke(D, r)$	$Read(r)$
		$Decrypt(r, K_{ADEF})$
		$Encrypt(r, K_{AEF})$
Batch revocation	$Revoke(\{B, C, D\}, r)$	$Read(r)$
		$Decrypt(r, K_{ABCDEF})$
		$Encrypt(r, K_{AEF})$

on a resource one by one and batching the revocations of the group of users on the resource. This is due to data resources with the same ACL encrypted with the same keys. We assume the header or the body of a data resource r is encrypted with a key K_{ABCDEF} at the surface encryption layer by the cloud, which means it can be read by a set of users {A,B,C,D,E,F} and now r just can be accessed by {A,E,F} after a series of revocations. We give a comparison between revoking {B,C,D} one by one and batching these revocations in Table 4. We can see that a reduction in the number of repetitive operations on the data resource by applying batch revocations. It can significantly lower much overhead for transmission and cipher operations, especially for the data resource of large size when re-encrypting the content in the body.

5.2 Security Analysis

Access control of sensitive data in our scheme is implemented by reasonably distributing keys of the two encryption layer (BEL and SEL). In the "Honest-but-Curious" model, protecting sensitive data against both unauthorized visitors and the cloud is difficult to implement when re-encryption for the update of access control policy relies on the cloud. Therefore, the security of our scheme lies in the distribution of the cryptographic keys over the two levels, which is executed by the data owner and the cloud provider by appropriately publishing public tokens to construct derivation paths.

In order to prevent sensitive data from unauthorized access, data resources are firstly encrypted with the data content key at the base encryption layer enforced by data owners. Adversaries must obtain the keys (data content key and base head key) of the base encryption layer in order to obtain the plaintext

of the data resource. As the data content key in the header is encrypted with the base head key, only with the base head key can the adversary decrypt the data content in the base encryption layer. In fact, the base head key in our approach is equal to the key at the BEL of over-encryption.

We adopt the cloud to protect the base head key of the header at the surface encryption layer. In fact, unauthorized users might obtain the base head key in our scheme. For example, a revoked user might locally maintain the base head key of the revoked resource; a newly granted user might acquire the base head key of the resource r_i unintendedly, when the user is authorized to access the resource r_j, which is encrypted with the same base head key of r_i. However, unauthorized users who have got the base head key cannot decrypt the data content because the cloud consolidates the defensive barrier. For those with just the base head key, the cloud encrypts the pre-encrypted data content key in the header with the surface head key. Adversaries cannot get the data content key without the surface head key generated by the cloud. For those who have got both the base head key and the data content key generated by the data owner (revoked users), the cloud encapsulates the data content by encrypting it with surface content key, and the surface content key is also protected by the surface head key. Adversaries cannot decrypt the data content without the surface head key. The surface head key is equal to the key to over-encrypt the pre-encrypted data content in the SEL of over-encryption.

Therefore, the security of our scheme lies in protecting the surface head key and the base head key, which equals to protecting keys at both the BEL and the SEL of over-encryption. The analysis of the related collusion attack by the cloud and the unauthorized users who have obtained the keys of the base encryption layer can be referred to over-encryption [4].

6 Related Work

In order to protect shared sensitive data from unauthorized access in incompletely trusted servers, shared cryptographic file systems which implement access control have obtained considerable development. SiRiUS [9] and Plutus [7] are earlier file systems, which adopt cryptographic techniques to implement access control. SiRiUS encrypts each data file and divides each file into a meta data file and an encrypted data content file, but the size of meta data file is proportional to the number of authorized users. Plutus groups different files and divides each file into multiple file blocks. Each file group uses a file lock box key and each file block is encrypted with a unique key. However, as different file groups attach different file lock box keys, maintaining multiple keys for a user is inadvisable.

Attribute-based encryption (ABE) which was first proposed in [10], is another branch to share sensitive data in the cloud environment without maintaining keys for each file or each file group. ABE is now widely researched in cloud computing to protect sensitive data [11–13]. Shucheng Yu presents a fine-grained data access control scheme in cloud computing [11], which combines ABE, proxy re-encryption [14,15] and lazy encryption. It supports policy update. However,

it cannot update a user' s privilege on a certain specific file, and revoking of users requires updating all the associated attributes and notifying the users who also maintain keys of the related attributes. Our approach just updates the key of the revoked resource.

Over-encryption [1,4] protects the shared sensitive data in "Honest-but-Curious" cloud and implements access control policy updates. Its architecture of access control is based on a key-derivation structure in [16], which is also adopted by [17–19]. In the key-derivation structure, a user just needs to maintain private keys to derive all the keys of the authorized resources. In the previous work of over-encryption, both granting and revoking need encrypt the related resources. This consumes a lot of resources and time, especially for those data of GB/TB/PB size.

To reduce the overhead of revocations, lazy revocation proposed in Cepheus [20] is widely adopted by existing cryptographic file systems [21]. Lazy re-encryption at the price of slightly lowered security [22] delays required re-encryptions until the next write access. Because it brings in much overhead for revocations (reading disc, decrypting data and encrypting data), we apply batch revocation combining lazy revocation, which reduces the overhead and improves the performance of the cloud storage service.

7 Conclusions

With the explosive growth of data, the outsourced data scale in the cloud will increase and be enlarged. However, security is the main obstacle in the way of outsourcing data to "Honest-but-Curious" cloud. Encrypting the outsourced data before uploading them to the cloud is a widely researched solution, but it brings new challenges to update the access control policy in order to share data. On the premise of implementing fine-grained access control, our scheme can achieve efficient updating of the access control policy in cryptographic cloud storage. The performance analysis shows that the proposed dual-header structure and batch revocation can significantly minimize the overhead for authorization. However, the collusion attack, launched by the cloud and the unauthorized users who have obtained keys of the base encryption layer, still cannot be solved in this paper. In order to alleviate the possibility of this collusion attack, dispersing data resources among multiple clouds and applying secret sharing techniques might be a selectable solution, which might be our next work. As the re-encryption on revoked resources is inevitable in almost all the cryptographic storage systems, efficient re-encryption on large data resource is also our next research direction.

Acknowledgment. This work is supported by by National 973 Program of China under award No. 2014CB340603.

References

1. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM Trans. Database Syst. (TODS) **35**, 12:1–12:46 (2010)
2. Liu, Q., Wang, G., Wu, J.: Time-based proxy re-encryption scheme for secure data sharing in a cloud environment. Information Sciences (2012)
3. Hohenberger, S.R., Fu, K., Ateniese, G., Green, M., et al.: Unidirectional proxy re-encryption. US Patent 8,094,810, 10 January 2012
4. Di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Over-encryption: management of access control evolution on outsourced data. In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 123–134, VLDB endowment (2007)
5. Kamara, S., Lauter, K.: Cryptographic cloud storage. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) RLCPS, WECSR, and WLC 2010. LNCS, vol. 6054, pp. 136–149. Springer, Heidelberg (2010)
6. Backes, M., Cachin, C., Oprea, A.: Lazy revocation in cryptographic file systems. In: Proceedings of IEEE Security in Storage Workshop (SISW 2005), pp. 1–11. IEEE (2005)
7. Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: scalable secure file sharing on untrusted storage. In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, vol. 42, pp. 29–42 (2003)
8. Resch, J.K., Plank, J.S.: AONT-RS: blending security and performance in dispersed storage systems. In: 9th Usenix Conference on File and Storage Technologies, FAST-2011 (2011)
9. Goh, E.-J., Shacham, H., Modadugu, N., Boneh, D.: SIRIUS: Securing remote untrusted storage. In: Proceedings NDSS, vol. 3 (2003)
10. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
11. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: INFOCOM, 2010 Proceedings IEEE, pp. 1–9. IEEE (2010)
12. Li, M., Yu, S., Ren, K., Lou, W.: Securing personal health records in cloud computing: patient-centric and fine-grained data access control in multi-owner settings. In: Jajodia, S., Zhou, J. (eds.) SecureComm 2010. LNICST, vol. 50, pp. 89–106. Springer, Heidelberg (2010)
13. Wang, G., Liu, Q., Wu, J.: Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 735–737. ACM (2010)
14. Ivan, A., Dodis, Y.: Proxy cryptography revisited. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2003)
15. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans. Inf. Syst. Secur. (TISSEC) **9**(1), 1–30 (2006)
16. De Capitani di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: A data outsourcing architecture combining cryptography and access control. In: Proceedings of the 2007 ACM Workshop on Computer Security Architecture, pp. 63–69. ACM (2007)
17. Raykova, M., Zhao, H., Bellovin, S.M.: Privacy enhanced access control for outsourced data sharing. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 223–238. Springer, Springer (2012)

18. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Support for write privileges on outsourced data. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IFIP AICT, vol. 376, pp. 199–210. Springer, Heidelberg (2012)
19. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Enforcing dynamic write privileges in data outsourcing. Comput. Secur. **39**, 47–63 (2013)
20. Fu, K.E..: Group sharing and random access in cryptographic storage file systems. Ph.D. thesis, Massachusetts Institute of Technology (1999)
21. Zarandioon, S., Yao, D.D., Ganapathy, V.: K2C: cryptographic cloud storage with lazy revocation and anonymous access. In: Rajarajan, M., Piper, F., Wang, H., Kesidis, G. (eds.) SecureComm 2011. LNICST, vol. 96, pp. 59–76. Springer, Heidelberg (2012)
22. Grolimund, D., Meisser, L., Schmid, S., Wattenhofer, R.: Cryptree: A folder tree structure for cryptographic file systems. In: 25th IEEE Symposium on Reliable Distributed Systems, SRDS 2006, pp. 189–198. IEEE (2006)