ICIC Express Letters
Volume **4**, Number **5**, October **2010**

 ISSN 1881-803X
pp. 1–ISII2010-000

A HIERARCHICAL DATA ACCESS AND KEY MANAGEMENT IN CLOUD COMPUTING

Tsuei-Hung Sun[1] and Min-Shiang Hwang[2]

[1]Department of Management Information Systems
National Chung Hsing University
250 Kuo Kuang Road, Taichung, Taiwan 402, R.O.C.
njpth24121@gmail.com

[2]Department of Computer Science and Information Engineering
Asia University
500 Liufeng Road, Wufeng, Taichung, Taiwan 402, R.O.C.
mshwang@asia.edu.tw
(Corresponding Author: Prof. Min-Shiang Hwang)

Received February 2010; accepted April 2010

Abstract. *Since cloud storage becomes a major application in cloud computing, many researches have been focusing on how to protect and verify the data stored in the Cloud. There is a problem that the owner doesn't want the real data stored by the Cloud to be Cloud known. Therefore, the most popular way is to encrypt entire file by using a secret key like Amazon Simple Storage Service (S3)* [1]*; however it may bring some problems such as secret key change and distribution, key management, and access rights management. So, we propose a hierarchical key management scheme to solve the problems mentioned above. And this scheme is suitable for individuals or enterprise service providers to manage their keys of files, software, and services.*
Keywords: Cloud computing, Access control, Key management, Over-encryption, Hierarchy.

1. **Introduction.** Cloud computing provides many advantages for individuals and enterprise service providers; for example, they don't need to worry about hardware such as storage, computing ability, and network connection quality [2]. Hence, providers can concentrate on developing software and providing better service for customers. However, when all files and software are outsourced to the Cloud, there will bring some new problems of data protection and verification. For example, the data stored in Cloud will be shared with others, but Data Owner doesn't want to let Cloud know what the real data is.

The most popular way is to use a secret key to encrypt the entire data before uploaded [3], and then to distribute the secret key to the access persons. But, this way is not secure enough in practice. In order to achieve data confidentiality, authentication and access control, there are some schemes using over-encryption with Access Control Lists (ACLs) [4] and Capability List (CapList) [5] to achieve the above goal. However, when Data Owner want to revoke the access right of some people, the secret key of the data need to be changed and resent to other related users.

However, these schemes [4, 5] cannot achieve changing key efficiently without affecting other users. On one hand, Data Owner might not retrieve the original data if the secret key is lose because Data Owner does not always keep the backup data in cloud storage scenario. On the other hand, since users need to access data from different Data Owners, users need to keep as many keys as many Data Owners. These two problems are so important not to be ignored in the environment of cloud computing.

Therefore, we propose a scheme that Data Owner can derivative the encrypt key of each data by using one own key; and when the encrypt key of specific data need to be changed, Data Owner can change it without affecting most related users. Some data access criteria needed to achieve in cloud computing are listed in the following:

1. Data Confidentiality: Data Owner needs to encrypt all data before data is uploaded to Cloud because the server of Cloud is not always worthy of be trusted.
2. Access Control: Data can only be downloaded and decrypted by the specific user who has the access right.
3. Authorization: Data Owner can add or revoke access right of a specific user, and change the encrypt key of the data without affecting most related users who have the access right of the data.
4. Key Management: Data Owner can use one key to derivative all encryption key of data without leaking any information related to other data; and users can use one key to derivative all data which he/she has the access right.

The rest of parts are composed as follows: Section is the related work about the over-encryption, capability list, and shared-key tree. In Section 3, we list the assumption, notation table, and our proposed scheme. Then discussion and analysis of the security, performance, and the comparison with the proposed scheme are in Section . In the end at Section is our advantages, drawbacks, and conclusion.

2. **Related Work.**

2.1. **Over-Encryption.**

Over-Encryption [4] is a scheme that protects the outsourced data; and this framework is suitable for cloud computing environment. Over-Encryption using two layers of encryption to let data can be managed by Data Owner and Cloud. The first layer at Data Owner side will encrypt the data before uploaded to Cloud, and maintain an encrypted data list and related user list for Cloud to manage the uploaded data. The second layer is at Cloud side, Cloud will encrypt the encrypted data before sent to users, which is used at the traditional server.

2.2. **Shared Key Tree.**

The Shared Key Tree is a scheme that can let users of the same access right being grouped together as a group (call the key tree), and share a group secret key to retrieve the encryption key of data wanted [6]. As shown in Figure 1, Data Owner will use the DEK to encrypt the data. Then, according to the access right to generate different key trees (each triangle) for the same access right user. The KDKs, the distribution keys, have the property of binary tree to decrease the affected users during key change. At the leaf of tree is the individual key (IDK) for each user.

However, we can find that if a user has many access rights of different data, user need to keep many keys (see Figure 1(a)). By shared the key tree (see Figure 1(b)), the user can just use one secret key to access many different data.

In summary, we find that the original over-encryption is not enough for Data Owner because the key management is complicated. Another problem is that all related keys need to be regenerated and distributed to all related users after a user leaves the system. This way may become a big overhead for both Data Owner and all related users. For that reason, we propose a more efficient scheme to improve the key management by combining the Shared Key Tree with Over-Encryption. There has more description at Section .

3. **The Propose Scheme.**

In this section, we will introduce assumptions and notations used in this scheme. Then, we describe the structure of our protocol. In the end are four phases of our scheme. Here are assumptions and restrictions of our scheme:

1. The server of Cloud needs to be always online, but Data Owner and users don't need to be always online in the cloud computing environment [5].

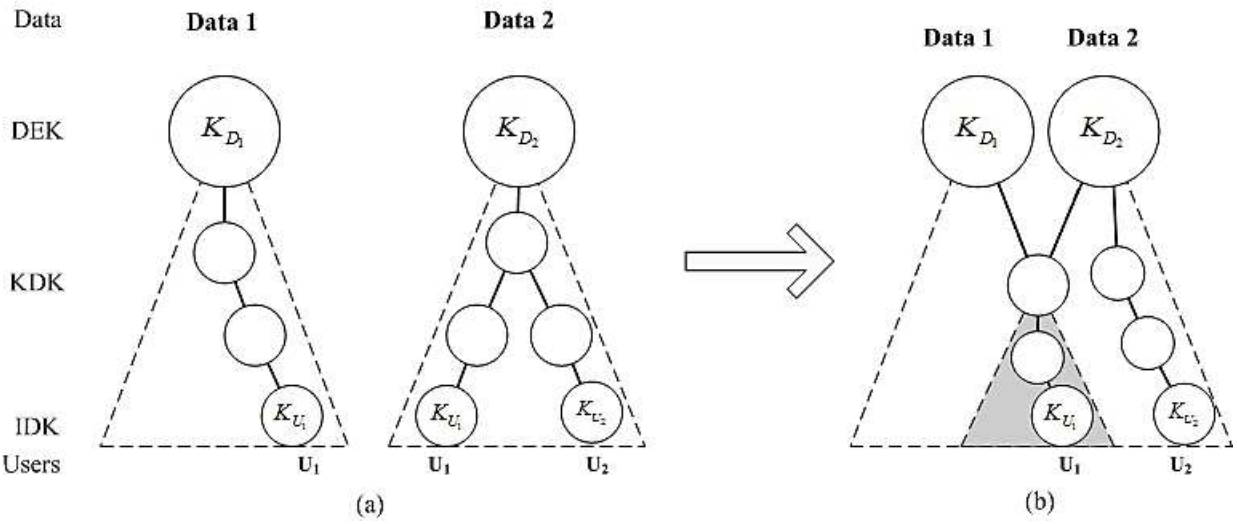

FIGURE 1. Shared Key Tree (SKT). (a) Not shared with others. (b) Shared with others.

2. The proposed scheme is more suitable at the scenario that the same file will be shared with many people.
3. When the user number is more than the data number at one shared key tree, the proposed scheme can achieve the largest benefit.

3.1. **Notation.** The notation table (see Table 1) lists all notations used in the proposed scheme.

TABLE 1. Notation Table

Notation	Significance
PU_X	X's public key, X is Data Owner(O), Cloud(C), or User(U).
PV_X	X's private key, X is Data Owner(O), Cloud(C), or User(U).
K_O	Data Owner's master key using to generate encrypt key of file.
K_U	User's master key that using to derivative encrypt key of file.
ARL	Access Right List of all user
UID	User's ID
FID_i	ID of file i, where $i = 1, ..., n$.
c_{F_i}	A counter of the number of change key of the file i
K_{F_i}	Encrypt key of file i, where $i = 1, ..., n$.
K_{r_i}	Root key of the group i, where $i = 1, ..., n$.
K_{t_i}	Tree key of the group i, where $i = 1, ..., n$.
T	Time stamp
EK	Encrypt function

3.2. **Our Scheme.** Here we find that Sanka et al.'s [5] scheme is a secure and simple structure for storage data in Cloud. However the scheme is too generalize to be used in all scenarios about storage in Cloud, and it is not very suitable for outsourcing data in practice. The outsourced data may be shared with many different people who may access many different data from the Cloud. However, Sanka et al.'s scheme need to keep each encrypt key of each data that is inconvenient for both Data Owner and users. So, we base on Sanka et al.'s scheme to let it more suitable for the scenario of outsourcing data.

In the proposed scheme, there have tree participants: Data Owner, Cloud, and User. Data Owner can be individuals or enterprise data providers that will upload the file to

Cloud to be shared with some people restrictedly. Cloud is a storage provider that will handle the data management and access control. And User may pay or get the data access right before using this system.

3.2.1. *Setup.* Data Owner generates a secret master key K_O and the encryption key of file $K_{F_i} = h(K_O \| FID_i \| c_{F_i})$. Then, Data Owner use K_{F_i} to encrypt file F_i and record the times of renewing the key. According to the access right, Data Owner will build the shared key tree (see Figure 2) and create a list ARL that records the association of User id UID and the encrypted data $EK_{K_{F_i}}(F_i)$ that the user has access right. Therefore, Data Owner will send $EK_{PU_C}(EK_{PV_O}(\{FID_1 \| EK_{K_{F_1}}(F_1), ..., FID_n \| EK_{K_{F_n}}(F_n)\}, ARL))$ to Cloud, Cloud decrypts the received massage and stores the encrypted file and ARL.

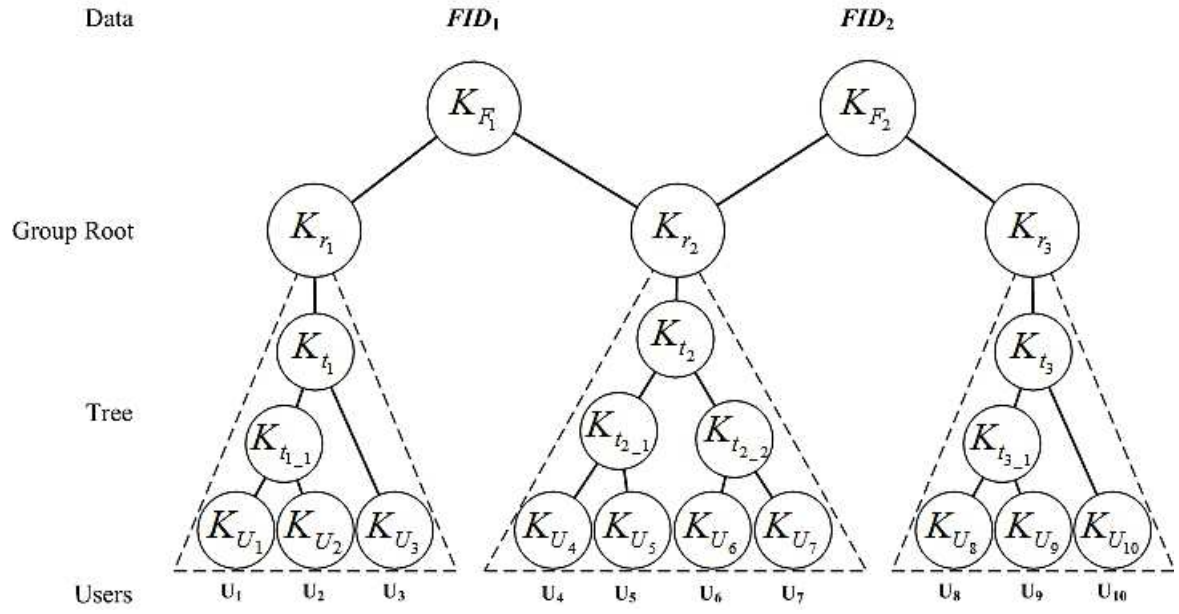

FIGURE 2. An Example of Shared Key Tree.

3.2.2. *Registration.* User sends $EK_{PU_O}(EK_{PV_U}(UID, \{FID\}, T))$ to Data Owner to ask the access right (the request can include many file by add FID). Data Owner first checks the User's request, and then adds UID and $FID_i \| EK_{F_i}(F_i)$ to ARL. Then, Data Owner will generate the K_U for User and add User to the shared key tree. Finally, Data Owner sends renewed key messages $EK_{PU_C}(EK_{PV_O}(FID, EK_{K_{r_i}}(K_{F_i}), EK_{K_{t_i}}(K_{r_i}), EK_{K_U}(K_{t_i})))$ to Cloud and $EK_{PU_U}(EK_{PV_O}(UID, K_U, T))$ to User. Cloud decrypt message and store FID with $EK_{K_{r_i}}(K_{F_i}), EK_{K_{t_i}}(K_{r_i}), EK_{K_U}(K_{t_i})$ to provide renewed key messages to User. User will decrypt and store K_U for decrypt renewed message and get K_F to decrypt the encrypted file.

3.2.3. *Download Data.* User sends $EK_{PU_C}(EK_{PV_U}(UID, FID, T))$ to Cloud, then Cloud will check ARL and renew key message. If there has the associated renewed key messages, Cloud will encrypt the encrypted $EK_{F_i}(F_i)$ with renewed key messages $EK_{K_{r_i}}(K_{F_i})$, $EK_{K_{t_i}}(K_{r_i})$, $EK_{K_U}(EK_{K_{t_i}})$ as $EK_{PU_U}(EK_{PV_C}(EK_{F_i}(F_i), EK_{K_{r_i}}(K_{F_i}), EK_{K_{t_i}}(K_{r_i}), EK_{K_U}(K_{t_i})))$ and send to User. Then, User can use K_U to decrypt the renewed messages one by one to get the encrypt file key K_F. Finally, User can use K_F to decrypt and get the original file.

4. **Discussion and Analysis.**

4.1. **Security Analysis.** Here we will analyze data confidentiality, access control, authorization, and Key Management of the proposed scheme that can achieve the above listed criteria.

Data Confidentiality: Through the over-encryption, data is encrypted by Data Owner before uploaded to the Cloud to avoid the curiosity of internal members in Cloud. And Cloud will encrypt the encrypted data before data is sent to User to avoid the attack of persons not allowed to intercept and decrypt it from the open channel.

Access Control: Each User has a unique master key which is on behalf of the User's specific access right. Only the User who has the access right can request the data from the Cloud and retrieve the original data by User's master key.

Authorization and Key Management: Data Owner can give the data access right to a specific person easily so to generate and add the User's master key to the shared key tree. Then, User can use the master key to retrieve the encryption key of the data. If Data Owner doesn't want to shard the data with some people, Data Owner just renews the encryption key of the data and sends the renewed key messages to the Cloud. Thus, other related remain users don't need to change their master key that can reduce the overhead of changing keys.

4.2. **Performance Evaluation.** The proposed scheme using the shared key tree is to achieve lowering overhead of changing encrypted key of data. As shown in Table 2, although User needs to operate more times of symmetrical decryption to decrypt the retrieval key, User only needs to remember one key K_U that is more convenient than to remember as many as the number keys K_{F_i} of F_i.

In addition, when Data Owner changes the encrypted key K_{F_i}, User doesn't need to change his/her master key and Data Owner doesn't need to resend the new key to all associated Users. The computing time of symmetrical encryption/decryption is much smaller then asymmetrical encryption/decryption, if a file need to change the encrypt key, Data Owner just need to operate 2 times asymmetrical encryption and $n + r$ times symmetrical encryption to generate renewed key message ones for all associated users, but Sanka et al.'s scheme need to operate 3 times asymmetrical encryption to re-generate decryption information for each associated user. If there has n users, our scheme just need send one message to Cloud and all related users can use it to decrypt file, but Sanka et al.'s scheme need to send n messages to Cloud.

Table 2. Comparison of Stored and Key Management

Cost	Our Scheme	Sanka et al. [5]
Stored		
Data Owner	ARL,K_O,K_{r_i},K_{t_j}	$CapList$,K_{F_i}
User	K_U	K_{F_i}
Cloud	ARL,$EK_{K_{F_i}}(F_i)$,$EK_{K_{r_i}}(K_{F_i})$, $EK_{K_{t_i}}(K_{r_i})$,$EK_{K_{U_i}}(K_{t_i})$	$CapList$,$EK_{K_{F_i}}(F_i)$
Key Management		
Change Key	renew K_{F_i} and associated K_{r_i},K_{t_i}	renewK_{F_i}
Affected User	No	all Users who has K_{F_i}
Lose Encrypt Key	recalculate by c_{F_i}	cannot recalculated
Computing Cost		
Data Owner	$(n+r)T_{EK_S}$+nT_h+$6T_{EK_{AS}}$ +$2T_{DK_{AS}}$	nT_{EK_S}+$7T_{EK_{AS}}$+$2T_{DK_{AS}}$
User	$4T_{EK_{AS}}$+$4T_{DK_{AS}}$+$(1+j)T_{DK_S}$	$3T_{EK_{AS}}$+$3T_{DK_{AS}}$+$2T_{DK_S}$
Cloud	$2T_{EK_{AS}}$+$4T_{DK_{AS}}$	$2T_{EK_{AS}}$+$6T_{DK_{AS}}$+$1T_e$+sT_e

Where T_h: the computing time of hash. T_{EK_S}: the computing time of symmetric encryption. $T_{EK_{AS}}$: the computing time of asymmetric encryption. T_{DK_S}: the computing

time of symmetric decryption. $T_{DK_{AS}}$: the computing time of asymmetric decryption. T_e: the computing time of exponentiation. n: the number of file. r: the number of renewed key. i: the number for User to decrypt the file. j: the number for User to decrypt the file and retrieval key. s: the number of Session.

5. **Conclusions.** In summary, we propose a more efficient scheme that can be used for Data Owner to manage all data easily, and change the encryption key for a user's access right in the condition without affecting the most related users in the system. And our scheme is more suitable for outsourcing data shared with other non-publicly. When the number of data is large and access users enter and leave dynamically, the Sanka et al.'s scheme may bring a very big obsession for both Data Owner and User since Data Owner needs to notice all associated users to renew the encrypted key and User is always required to change the encryption key of data.

There is a trade-off between the storage and computing time. In the cloud computing environment, both Data Owner and User want to access more data easily. If User needs to bring a lot of keys for accessing the data in Cloud, it is very inconvenient. And the other problem is that Data Owner may not store the data in his/her own computer. If Data Owner needs to remember a lot of keys, it will become a very big burden when the keys are lost. However, we use the master key with the file id and counter to generate the encrypted key of data to solve the problem. Even if Data Owner loses the encrypt key, the key can be regenerated by using the counter.

Acknowledgment. This work is partially supported by National Science Council under the grants NSC 98-2221-E-005-050-MY3. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] W. Wei, J. Du, T. Yu and X. Gu, SecureMR: A service integrity assurance framework for MapReduce, *Proc. of the 25th Annual Computer Security Applications Conference*, Honolulu, USA, pp.73-82, 2009.

[2] C. Wang, Q. Wang, K. Ren and W. Lou, Ensuring data storage security in cloud computing, *Proc. of the 17th International Workshop on Quality of Service*, South Carolina, USA, pp.1-9, 2009.

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, Above the clouds: A berkeley view of cloud computing, *No. UCB/EECS-2009-28, University of California at Berkley, USA*, pp.1-23, 2009.

[4] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati, Over-encryption: Management of access control evolution on outsourced data, *Proc. of the 33th International Conference on Very Large Data Bases*, Vienna, Austria, pp.123-134, 2007.

[5] S. Sanka, C. Hota and M. Rajarajan, Secure data access in cloud computing, *Proc. of the 4th Internet Multimedia Services Architecture and Application*, Bangalore, India, pp.1-6, 2010.

[6] Q. Gu, P. Liu, W.C. Lee and C.H. Chu, KTR: An efficient key management scheme for secure data access control in wireless broadcast services, *IEEE Trans. Dependable Secur. Comput.*, vol.6, no.3, pp.188-201, 2009.