A Time-Based Group Key Management Algorithm Based on Proxy Re-encryption for Cloud Storage

Yihui Cui, Zhiyong Peng, Wei Song[*,**], Xiaojuan Li, Fangquan Cheng, and Luxiao Ding

Computer School, Wuhan University, Wuhan, China
{cuiyihui,peng,songwei,lxj,cheng,dingluxiao}@whu.edu.cn

Abstract. Users are motivated to outsource their data into the cloud for its great flexibility and economic saving. However, outsourcingdata to cloud also increases the risk of privacy leak.A straightforward method to protect the users'privacy is to encrypt the files before outsourcing.The existing group key management methods always presume that the server is trustworthy, but cloud storage applications do not meet this condition. Therefore, how to manage the group key to enable authenticated usersto access the files securely and efficientlyis still a challenging problem.In our paper, wepropose a Time-basedGroup Key Management (TGKM)algorithmforcryptographiccloud storage applications, which uses the proxy re-encryption algorithm to transfermajorcomputingtask of the group key management to the cloud server.So, the proposed TGKM scheme greatly reduces the user's computation and storage overhead and makes full use of cloud server to achieve an efficient group key management for the cryptographic cloud storage applications.Moreover, we introduce a key seed mechanism to generate a time-based dynamic group key which effectively strengthens the cloud data security. Our security analysis and performance evaluations both show that the proposed TGKM scheme is a secure and efficient group key management protocol for the cloud storage applications with low overheads of computation and communication.

Keywords: cryptographic cloud storage,proxy re-encryption,group key management.

1 Introduction

Cloud storage is a typical service model of online outsourcing storage where data is stored in virtualized pools which are generally hosted by third parties. Companies need only pay for the storage they actually use. But when data is stored into cloud, user simultaneously loses the control of his data. It makes that the unauthorized accesses from hackers even cloud service providers is inevitable. Security is one of the most important problems that should be addressed in cloud storage applications [1].

[*] Corresponding author.

[**] This work is partially supported by National Natural Science Foundation of China No. 61202034, 61232002 and Program for Innovative Research Team of Wuhan(2014070504020237).

L. Chen et al. (Eds.): APWeb 2014, LNCS 8709, pp. 117–128, 2014.

In recent years, many scholars have proposed the use of encryption methods to protect users' privacy in cloud storage applications [2-6]. In cryptographic cloud storage application framework data owner encrypts files before outsourcing to protect his privacy. Because the authorized users have the key, they could decrypt the files after downloading. Obviously, unauthorized users, attackers, even the cloud service provider can't breach user's privacy without authentication. In cryptographic cloud storage, data owner need not only store files on the cloud but also shares these files to some group users. Therefore, group key management is an important problem in cloud storage, and it is also the main motivation of our paper.

The problem of group key management in cryptographic cloud storage environment is different from the traditional one. In a cryptographic cloud storage model, computing tasks should be transferred to the cloud as much as possible and ensure user privacy at the same time. The main contributions of our work are:

- We propose a suitable group key management method of cloud storage, which transfers calculations to the cloud computing service providers, who can't get the group key.
- The data owner and authorized group users compute different group keys in different phases with the same seed, rather than always using the same group key, so our method is safer. Besides, because group key in a phase is computed by key seed, the distribute group key number of times is less than traditional method

The remainder of this paper is organized as follows: in Section 2, we discuss the related work. Then we introduce several cryptographic primitives in Section 3. Section 4 details the TGKM. Security analyses of TGKM will be given in Section 5. Finally, we evaluate the performance of our mechanism in Section 6, and conclude this paper in Section 7.

2 Related Work

There are many group key management algorithms to address the problem of group key management in the network environments, some are depended on a trusted group key server, and others don't need any trusted group key servers.

Xiao proposes a cryptographic file system called CKS-CFS based on the security assumption that the CKS-CFS is trusted [7]. A trusted Group Key Server (GKS) is introduced to manage file encryption keys in a centralized manner and to enable the employment of flexible access control policies. But if GKS is invaded, hacker can get all the private files.

Goh proposes the SiRiUS which doesn't need a trusted group key server usually let each user has a public and private key pairs to obtain the group key [8]. When a data owner wants to share data, he uses the group key to encrypt the file and uses the authorized user's public keys to encrypt the group key, and then he uploads the encrypted file and encrypted keys to the cloud. The authorized user uses his private key to decrypt the group key by which the authorized user decrypts the encrypted file. This method is one of the simplest group key managements, but it requires that the

data owner encrypts the group key for each user using his public key, which will generate a great overhead of computing at the data owner. Kim proposes a secure protocol called Tree-based Group Diffie–Hellman (TGDH) that is both simple and fault-tolerant[9]. In order to protect the security of data, different files are encrypted by different keys. But the processes of key negotiation in TGDH need to replace the user's private key, so the algorithm is not suitable for group key management in cloud storage.

3 Preliminaries

3.1 Proxy Re-encryption

Proxy re-encryption schemes are cryptosystems which allow third-parties (proxies) to alter a ciphertext which has been encrypted for one party, so that it may be decrypted by another. However the third-parties can't get the secret value [10]. Blaze presents the BBS, Elgamal-based scheme operating over two group G_1,G_2 of prime order q with a bilinear map $e: G_1 \times G_1 \rightarrow G_2$. The system parameters are random generators $g \in G_1$ and $Z = e(g, g) \in G_2$.

- Key Generation(KG). The user A select random x∈Z_q. A's key pair is the form $pk_a = g^a$, $sk_a = a$.
- Re-Encryption Key Generation(RG). A user A delegates to B by publishing the re-encryption key $rk_{A\rightarrow B} = g^{b/a} \in G_1$, computed from B's public key.
- First-Level Encryption(E_1). to encrypt a message $m \in G_2$ under pk_a in such a way that it can only be the holder of sk_a, output c=(Z^{ak}, mZ^k) .
- Second-level Encryption(E_2). to encrypt a message $m \in G_2$ under pk_a in such a way it can be decrypted by user A and his delegates, output c=(g^{ak}, mZ^k).
- Re-Encryption(R). Anyone can change a second-level ciphertext for A into a first-level ciphertext for B with $rk_{A\rightarrow B}$. From $c_a = (g^{ak}, mZ^k)$, compute $e(g^{ak}, g^{b/a}) = Z^{bk}$ and publish $c_b = (Z^{bk}, mZ^k)$.
- Decryption(D_1, D_2) . To decrypt a first-level ciphertext $c_a = (\alpha, \beta)$ with secret key sk =a,compute $m = \beta/\alpha^{1/a}$. To decrypt a second-level ciphertext $c_a = (\alpha, \beta)$ with secret key sk=a, compute m=$\beta/e(\alpha, g)^{1/a}$.

3.2 Chinese Remainder Theorem

Suppose m_1,m_2, …, m_k are positive integers that are pairwise coprime. Then, for any given sequence of integers a_1, a_2, …,a_k, there exists an integer x solving the following system of simultaneous congruences[11].

$$\left.\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}\right\} \tag{1}$$

Furthermore, all solutions x of this system are congruent modulo the product $N=m_1m_2\ldots m_k$, so the value of x mod N is unique.

4 TGKM: A Time-Based Group Key Management Algorithm

Table 1 shows the notations in the following of this paper.

Table 1. Notations

Notations	Description
$GK_{file(j)}$	Files which upload in T_j is encrypted by $GK_{file(j)}$
S_{key}	Key seed S_{key}
$S_{forward(i)}$	Forward key seed in T_i is used to compute $K_{forward}$ in one phase
$S_{backward(i)}$	Backward key seed in T_i is used to compute $K_{backward}$ in one phase
$K_{forward(j)}$	Forward assistant key in T_j is used to compute GK_{file} in T_j
$K_{backward(j)}$	Backward assistant key in T_j is used to compute GK_{file} in T_j
$\{file\}_{GK_{file}}$	The file is encrypted by GK_{file}
$\{S_{key}\}PK_A$	The S_{key} is encrypted by user A's public key
$RK_{A\to B}$	The re-encryption key from A to B
T_j	The time phase in T_j
g	The system parameters are random generators $g \in G_1$

We design TGKM to implement an efficient and scalable group key management service for the cloud storage applications. The TGKM system model has three parties as follows:

(1) Data Owner: data owner encrypts his data and stores data in the cryptographic cloud storage system, and he not only uses data but also authorizes data to other user groups to access his data.

(2) Authorized Group Users: users who have the permission to access the encrypted data after authorized by the data owner to the group which the users belong to.

(3) Cloud Service Provider: the cloud offers data storage and sharing services to users. It follows our proposed protocol in general, but also tries to find out as much secret information as possible.

TGKM uses two steps to share the GK_{file} in the authenticated group users. GK_{file} is not fixed in various phases, so even GK_{file} is disclosed during any period, other GK_{file} is still secure.

In the first step, TGKM shares the key seed S_{key} based on proxy re-encryption mechanism in the authorized group users. We use the S_{key} to represent the key seed which consists of $S_{forward}$ and $S_{backward}$. The pair $\{S_{forward}, S_{backward}\}$ can compute file encryption group key. Then data owner and authorized users further compute time-based group keys from S_{key} to enable forward security and back ward security. Fig.1 describes TGKM for cryptographic cloud storage applications, and it is

composed of three parts: data owner domain, cloud domain, and authorized user domain. Data owner domain is a full trusted service domain in which data owner generates key seed S_{key} and uploads it to the cloud after encryption. The cloud domain is an untrusted service domain with powerful computing capability, TGKM introduces a proxy re-encryption tree structure to efficiently share key in the authorized group users by transferring data owner encrypted S_{key} to the key seed encrypted by authorized group users. In TGKM structure, the authorized users in the authorized group user domain could download the transferred encrypted S_{key} and decrypts it by his private key.

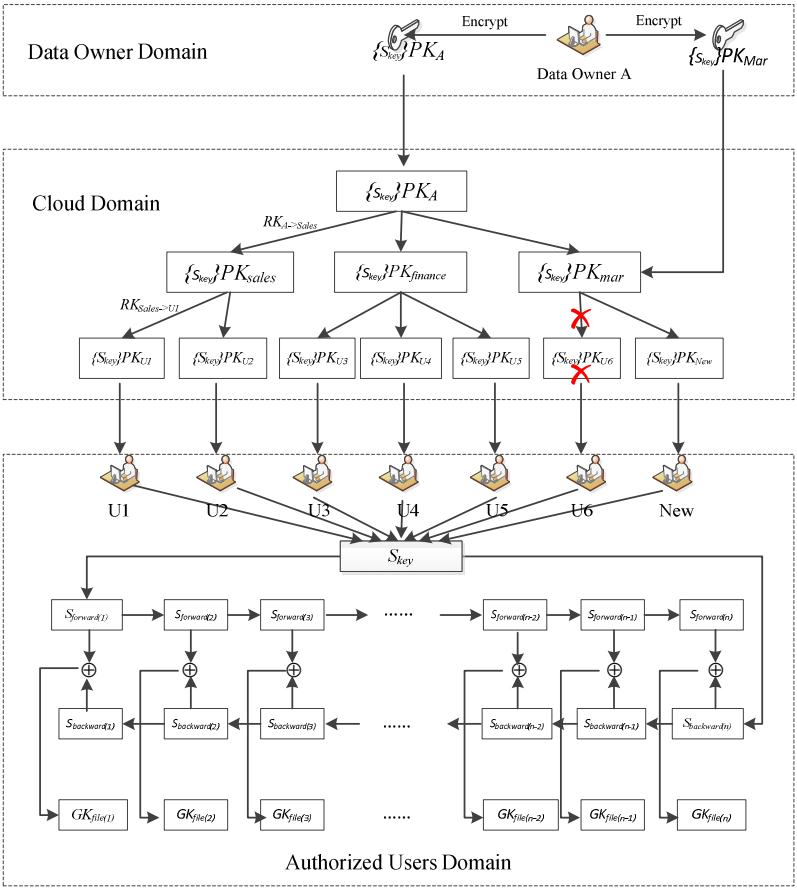


Fig. 1. Time-based Group Key Management for Cryptographic Cloud Storage (TGKM)

In the second step, as shown in the Fig. 1, the authorized group users get a set of keys $S_{forward(1)}$ and $S_{backward(n)}$ from the key seed S_{key}. In TGKM model, every authorized user group builds a hash function link to compute the GK_{file}. For example in Fig. 1, U_1 gets S_{key} which includes $S_{forward(1)}$ and $S_{backward(k)}$ from cloud, and U_1 can compute $S_{forward(i+1)}$ from $S_{forward(i)}$ and compute $S_{backward(i-1)}$ from $S_{backward(i)}$. And then

U_1 can get all the pair keys $S_{forward(i)}$ and $S_{backward(i)}$ ($1 \le i \le n$) based on which further to get $GK_{file(i)}$ through $GK_{file(i)} = S_{forward(i)} \oplus S_{backward(i)}$. So, by this mechanism the key GK_{file} in any period is determined and enable a time-based key shared to achieve the forward security and backward security. For the data owner and the authorized group users can compute the same $S_{forward(i)}$ and $S_{backward(i)}$, they can share the same $GK_{file(i)}$ of any phase.

In this work, we just consider honest but curious cloud servers as [2] does. That is to say, cloud servers will follow our proposed protocol in general, but try to find out as much secret information as possible based on their inputs.

4.1 Cryptographic Cloud Storage Initialization Processes

During initial processes, the cryptographic cloud storage server generates the system parameters which include a random generators $g \in G_1$, $Z = e(g, g) \in G_2$, and m_1, m_2 which are two positive pairwise coprime integers.

4.2 User Basic Operations

- ***Register a User A.*** The user A gets the system parameters from cloud server first of all, and generates a random number $\alpha \in Z_P^*$ as A's private key SK_A. Then A generates his public key PK_A=[g, h=g^{α}] and uploads PK_A to cloud to finish registration.
- ***Create a Group.*** The data owner generates a random number $\beta \in Z_P^*$ as the group private key. Then he generates $PK_{group} = [g, h = g^{\beta}]$ as the public key of the group. Finally, he computes the re-encryption key $RK_{A \to group} = \beta / \alpha$ in which α is the private key of data owner and uploads it to the cryptographic cloud storage server. For example in Fig.1, the data owner creates three authorized groups including: sales group, finance group, and market group.
- ***Authorize a User.*** The data owner A authorizes a user B and put it into certain group. A gets B's public key $PK_B = [g, h = g^{\gamma}]$ from cloud. And then A computes re-encryption key $RK_{group \to B} = g^{\gamma/\beta} \in G_1$ and uploads it to the cryptographic cloud storage server.
- ***Revoke an Authorized User.*** The data owner A requests cloud server to delete the re-encryption key of the revoking user. In Fig.1 we can see that the cloud server deletes the edge from PK_{mar} to U_6 to revoke U_6's privilege.
- ***Build the Key Management Structure in Cloud.*** Cryptographic cloud storage server builds the authorized tree to share resources in the authorized users. As is shown in cloud domain in Fig.1, each data owner has an authorized tree to describe the shared relationship of his resources. In the authorized tree, the root node stores key seed which is encrypted by the data owner's public key, each the child node presents a user group which stores key seed encrypted by the group public key, and every leaf node presents an authorized user stores the key seed encrypted by user's public key. And the edges describe the re-encryption operations and store the proxy re-encryption key.

4.3 The First Step of TGKM: Key Seed Distribution

The main motivation of our paper is to enable a time-based access control for the cloud storage applications. We introduce a key seed mechanism to achieve it. In this section, we introduce the key seed distribution.

The data owner *A* encrypts a key seed S_{key} under PK_A in such a way it can be decrypted by *A* and his delegates. *A* uploads $\{S_{key}\}PK_A$ to the cloud server.

$$\{S_{key}\}PK_A=\left(g^{\alpha k}, S_{key}Z^k\right), \tag{2}$$

The TGKM cloud server masters proxy re-encryption key $RK_{A\to group}=\beta/\alpha$ and $RK_{group\to B}=g^{\gamma/\beta}$ to distribute key seeds in all the authorized groups and their users, so that the cryptographic cloud server can transfer the key seed encrypted by data owner to the key seed encrypted by authorized group public key. The re-encryption from *A* to a group is described in equation (3) and (4) in which g^{β} is the group's public key. The transfer from a group to a user is shown in (5) and (6).

$$g^{\beta k}=(\mathrm{g}^{\alpha \mathrm{k}})^{\beta/\alpha} \tag{3}$$

$$\{S_{key}\}PK_{group}=\left(g^{\beta k}, S_{key}Z^k\right), \tag{4}$$

$$e\left(g^{\beta k}, g^{\gamma/\beta}\right)=Z^{\gamma k} \tag{5}$$

$$\{S_{key}\}PK_{user}=\left(Z^{\gamma k}, S_{key}Z^k\right)\ . \tag{6}$$

Through the above re-encryption, the authorized group user can decrypt key seed from key seed encrypted by the data owner. The decryption is illustrated in equation (7) in which γ is authorized user's private key.

$$S_{key}=S_{key}Z^k/\left(Z^{\gamma k}\right)^{1/\gamma}, \tag{7}$$

After getting the key seed, the user can compute GK_{file} by $K_{forward}$ and $K_{backward}$ which is generated by the key seed. For example in Fig.1, the data owner generates $\{S_{key}\}PK_A$ and uploads it to the cloud server. The cloud server transfers $\{S_{key}\}PK_A$ to $\{S_{key}\}PK_{sales}$ by $RK_{A\to sales}$, then transfers $\{S_{key}\}PK_{sales}$ to $\{S_{key}\}PK_{U1}$ by $RK_{sales\to U1}$.As a result, the user U_1 can decrypt $\{S_{key}\}PK_{U1}$ to get S_{key}.

If the data owner just grants an encrypted file's accessing privilege to a group, he encrypts the key seed S_{key} by the group public key. Such as in the Fig.1, the data owner only allows the market group to access an encrypted file, and then he encrypts the seed by the market group public key, and uploads$\{S_{key}\}PK_{Mar}$ in which PK_{Mar} is the public key of group market to the cloud.

4.4 The Second Step of TGKM: Computing GK_{file} by Key Seed

In our TGKM scheme, we introduce key seed S_{key} to enable the efficient and flexible time-based access control. The file encryption key management in TGKM is a time-based dynamic key which uses different key to encrypt files in different period.

The data owner generates Key seed $\mathrm{S_{key}}$ which consists of a forward seed $S_{forward}$ and a backward seed $S_{backward}$. The prior $S_{forward}$ can compute the next

$S_{forward}$ by a hash function. For the same reason, the behind $S_{backward}$ can compute the prior $S_{backward}$ by another hash function.

$$\left.\begin{aligned} S_{forward(i+1)} &= f_{forwardhash}\ (S_{forward(i)}) \\ S_{backward(i)} &= f_{backwardhash}\ (S_{backward(i+1)}) \end{aligned}\right\} \tag{8}$$

For example, in Fig.2 If data owner wants to limit a group user accessing the uploaded files from T_1 to T_3, The key seed he distributed is $\{S_{forward(i)},\ S_{backward(i)}\}$. If data owner wants to limit a group accessing to the uploaded files from T_1 to T_6, The key seed he distributed is $\{\ S_{forward(i)},\ S_{backward(i+1)}\}$.

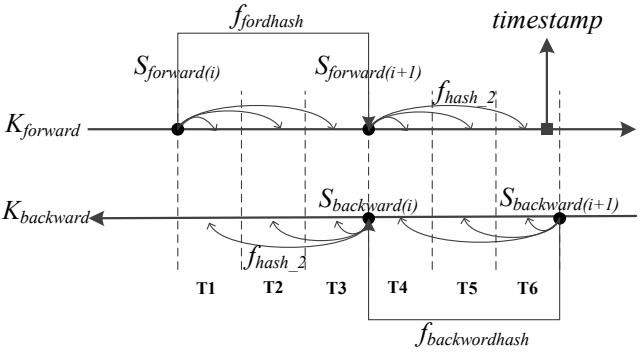


Fig. 2. Key seed mechanism to enable time-based data access control

Data owner and authorized group users use $S_{forward(i)}$ and $S_{backward(i)}$ to compute $K_{forward(j)}$ and $K_{backward(j)}$ separately by hash function (9). Every $S_{forward(i)}$ and $S_{backward(i)}$ can compute $K_{forward(j)}$ or $K_{backward(j)}$ of one time period. T_j is the time phase.

$$\left.\begin{aligned} K_{forward(j)} &= f_{hash\_2}\ (S_{forward(i)}, T_j) \\ K_{backward(j)} &= f_{hash\_2}(S_{backward(i)}, T_j) \end{aligned}\right\} \tag{9}$$

In Fig.2, $S_{forward(i)}$ can compute $K_{forward}$ from T_1 to T_3. $S_{backward(i)}$ can compute $K_{backward}$ from T_1 to T_3. The data owner and authorized group users shared m1 and m2 which meets $gcd(m_i, m_j) = 1$ in formula (10). When they have the same $K_{forward(j)}$ and $K_{backward(j)}$ they can generate the same $GK_{file(j)}$ based on Chinese remainder theorem.

$$\left.\begin{aligned} GK_{file(j)} &\equiv K_{forward(j)}\ (mod\ m_1) \\ GK_{file(j)} &\equiv K_{backward(j)}(\ mod\ m_2) \end{aligned}\right\} \tag{10}$$

Data owner and authorized group users can get the same $\{\ S_{forward(i)}\ ,\ S_{backward_{(i)}}\}$, so they can compute the same file encrypted group key. In Fig.2 user can generate $K_{forward(j)}$ in every phases by the forward seed $S_{forward(i)}$, and back forward seed $S_{backward(i)}$ can generate $K_{backward(j)}$, and then generate the $GK_{file(j)}$ at a certain period.

Algorithm 1. Compute file encryption group key $GK_{file(j)}$ by S_{key}

Input: $m_1, m_2, S_{forward(i)}, S_{backward(i)}, T_j$

Output: File encryption key $GK_{file(j)}$

1. $K_{forward(j)} = f_{hash\_2}(S_{forward(i)}, T_j)$
2. $K_{backward(j)} = f_{hash\_2}(S_{backward(i)}, T_j)$
3. $C \leftarrow 1.$
4. $u \leftarrow m_1^{-1} \bmod m_2.$
5. $C \leftarrow u \times C \bmod m_2.$
6. $u \leftarrow K_{forward(j)}, x \leftarrow u.$
7. $u \leftarrow (K_{backward(j)} - x)C \bmod m_2, x \leftarrow x + um_1,\ GK_{file(j)} = x$
8. Return $GK_{file(j)}$

4.5 Data Sharing to Group Users

When the data owner wants to upload a shared file, he gets the current time as the *timestamp* and computes encrypted file group key $GK_{file(j)}$ in the time phase by key seed, and encrypts file by $GK_{file(j)}$. Finally, the data owner uploads the encrypted file and the timestamp to the cloud server.

$$Files_{upload} = [\{file\}_{GK_{file(j)}}, timestamp] \tag{11}$$

When an authorized group user attempts to access a file, he firstly downloads $[\{file\}_{GK_{file(j)}}, timestamp]$. After downloaded, he computes encrypted file key $GK_{file(j)}$ by the algorithm 1 and decrypts $\{file\}_{GK_{file(j)}}$.

For example in Fig. 2, when data owner uploads a file, he gets the current time as timestamp. Then the data owner determines that accessing time phase is T_6. Finally, he computes the $GK_{file(6)}$ by $S_{forward(i+1)}$ and $S_{backward(i+1)}$. The authorized user can get the timestamp from $Files_{upload}$, and he determines that time phase is T_6. Consequently, he can get the $GK_{file(6)}$ by $S_{forward(i+1)}$ and $S_{backward(i+1)}$ to achieve the time-based accessing.

5 Security and Performance Analysis

5.1 TGKM Correctness Guarantee

Because the data owner and the authorized group share the same timestamp, they can determine the time phase which the timestamp is belonged to. They also get the same pair $\{S_{forward(i)}, S_{backward(i)}\}$, so they can compute the same pair $\{K_{forward(j)}, K_{backward(j)}\}$. According to Chinese remainder theorem, the data owner and authorized group users can get the consistent $GK_{file(j)}$ by formula (10). By the TGKM mechanism, authorized user can only get the corresponding seed key which is generated by the data owner according to his own will. By the seed key mechanism, we achieve a time-based access control to limit all the authorized users accessing data in the period of time defined by the data owner.

5.2 Forward Security and Backward Security Guarantee

There are two types of security requirements on a secure group key management system: the **forward security** and the **backward security**. The former refers to a newly joined user cannot gain access to the past group keys. And the latter refers to after a user has left the secure group, he should not be able to gain access to the future group keys [14]. The proposed TGKM can fully meet the forward security and the backward security. The forward security requires that the authorized group user can't access any file encryption group key $GK_{file(j)}$ before start time of key seed. This notion was first proposed by Günther[15]. The backward security requires that a revoked user can't access file encryption group key $GK_{file(j)}$ after end time of key seed. In our key seed structure, the authorized user only knows $\{S_{forward(i)}, S_{backward(k)}\}$ ($i<=k$), so he can only obtain $K_{forward}$ and $K_{backward}$ from phase i to k, that is to say he just can compute file encryption group key GK_{file} from phase i to k. Because the front forward key seed $S_{forward(i)}$ can compute the back forward key seed $S_{forward(i+1)}$ by hash function, but a posterior forward key seed $S_{forward(i+1)}$ can't compute the prior forward key seed $S_{forward(i)}$. It is as the same to the backward key's computation.

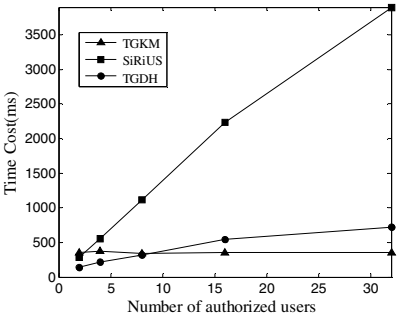
5.3 Computing Overhead Analysis

In TGKM, most of group key management computing operations is transferred to cloud. For computing $GK_{file(j)}$ by key seed, the main computing overhead is to compute $GK_{file(j)}$ by Chinese remainder theorem, so the time complexity is almost linear. And the computing overhear of cloud sever is O(h) in which h is the number of authorized users. As the TGKM algorithm mentioned above, the computing overhead at the user is related to the time of computing GK_{file} by forward and backward seed keys, so it is O(*1*). In the experimental section, we will carry out experiments to evaluate the performance of TGKM's efficiency.

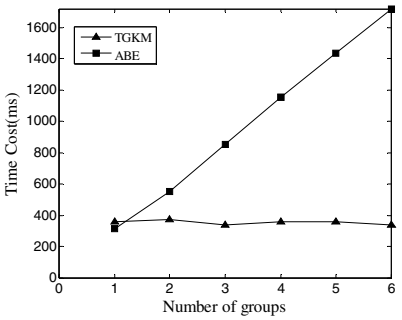
6 Experiments

In this section, we carry out experiments to evaluate the performance of proposed TGKM. All the experiments are executed under Ubuntu with an Inter Pentium 2.1GHz Processor and 1GB memory. The re-encryption algorithm is used JHU-MIT Proxy Re-cryptography Library [12]. We evaluate the efficiency of TGKM including: cost of distribution the keys at the data owner, computing overhead on the cloud server and cost of getting the keys at the client.

In the time cost experiments of data owner distributing the keys, we compare TGKM to SiRiUS, TGDH, and ABE. The experimental results are illustrated in the Fig. 3. As the results shown in the Fig.3 (a), the distributing group key cost on the data owner of SiRiUS and TGDH both rise with users' size increasing, while TGKM's time cost is almost not changed. Analyzing this phenomenon, we find that TGDH is a tree-based group key management algorithm which makes the tree layer increasing with the number of users increasing, so that key negotiation time increases as well. It is also in agreement with the experimental results in Fig. 3(a). However, TGKM data owner only encrypts key seed once based on the group, so the time cost of TGKM will not rise with user increasing. ABE is an



(a) with the change of number of users



(b) with the change of number of groups

Fig. 3. Time cost of data owner distribution group key

efficient find-grained authentication method which can be a method of group key management by treating a group as an attribute [13]. The experiments in Fig.3(b) shows the time overhead of TGKM and ABE. With the number of user group increasing, our proposed TGKM achieve a better computing performance than ABE.

Fig.4 shows the time cost of authorized user' computing the group key. The time cost of TGKM and SiRiUS is approximately equal. The time cost is decrypt group key by authorized user private key. And the time cost of TGKM is almost not changed with authorized user number increasing. But the time cost of TGDH is increase with authorized user number increase. The authorized user needs to negotiate with other group user.

Fig.5 shows cloud computing time cost. The overload is transfer from data owner to cloud. To the first group user, cloud has two proxy re-encryption operations. To the other users of the same group, cloud just has one proxy re-encryption operation.

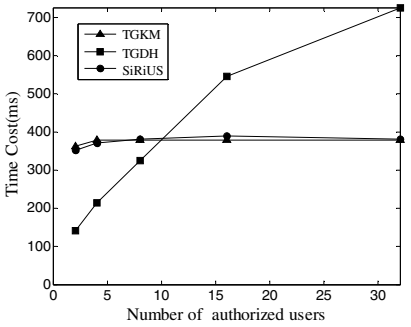


Fig. 4. Time cost for a user to get his key

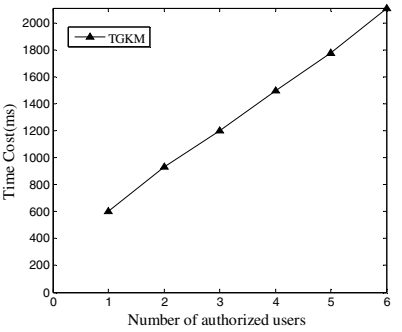


Fig. 5. Cloud computing time

7 Conclusion

When enterprises or individuals use cryptographic cloud storage applications to outsource their sensitive data, how to efficiently share data in the authorized group users without privacy leak is still one of the most challenging tasks. In this paper, we propose a novel time-based group key management (TGKM) in cryptographic cloud

storage. TGKM transfers much workload of key management to the cloud and prevents the cloud to master any group key. Furthermore, to enhance the scalability of TGKM with dynamic group, we propose the key seed mechanism to enable a time-based key management. Even if an attacker gets a file encryption key GK_{file}, he still can't decrypt any other files out of the time window. Through experiments, we find TGKM can greatly improve the efficiency of key management and can be applied to the cryptographic cloud storage applications.

References

1. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data. IEEE Trans. Parallel Distrib. Syst. 25(1), 222–233 (2014)
2. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable and fine-grained data access control in cloud computing. In: Proceedings of IEEE INFOCOM 2010, pp. 15–19 (2010)
3. Wang, Q., Wang, C., Ren, K., Lou, W., Li, J.: Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing. IEEE Trans. Parallel Distrib. Syst. 22(5), 847–859 (2011)
4. Kamara, S., Lauter, K.: Cryptographic Cloud Storage. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) FC 2010 Workshops. LNCS, vol. 6054, pp. 136–149. Springer, Heidelberg (2010)
5. Hong, C., lv, Z., Zhang, M., Feng, D.: A Secure and Efficient Role-Based Access Policy towards Cryptographic Cloud Storage. In: Wang, H., Li, S., Oyama, S., Hu, X., Qian, T. (eds.) WAIM 2011. LNCS, vol. 6897, pp. 264–276. Springer, Heidelberg (2011)
6. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Over-encryption: Management of access control evolution on outsourced data. In: Proc. of VLDB 2007, Vienna, Austria (2007)
7. Xiao, D., Shu, J.-W., Xue, W., Liu, Z.-C., Zheng, W.-M.: Design and implementation of a group key server-based cryptographic file system. Chinese Journal of Computers 31(4), 600–610 (2008)
8. Goh, E.-J., Shacham, H., Modadugu, N., Boneh, D.: SiRiUS: Securing Remote Untrusted Storage. In: NDSS 2003 (2003)
9. Kim, Y., Perrig, A., Tsudik, G.: Tree-based group key agreement. ACM Trans. Inf. Syst. Secur. 7(1), 60–96 (2004)
10. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. In: NDSS 2005 (2005)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The Chinese remainder theorem, sec.31.5, pp. 873–876. MIT Press and McGraw-Hill (2001) ISBN 0-262-03293-7
12. `http://spar.isi.jhu.edu/~mgreen/prl/`
13. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-Policy Attribute-Based Encryption. In: 28th IEEE Symposium on Security and Privacy 2007, pp. 321–334 (2007)
14. Yang, Y.R., Lam, S.S.: A Secure Group Key Management Communication Lower Bound, University of Texas at Austin, Austin, TX (2000)
15. Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 29–37. Springer, Heidelberg (1990)