ISSN 2277-3061

A Group Collaboratable Proof of Retrievability Scheme for Cloud Data Storage

Jyh-Shyan Lin[1], Kuo-Hsiung Liao[2], Chao-Hsing Hsu[3]
[1]Department of Information Management, Yuanpei University, No.306, Yuanpei St., HsinChu, Taiwan, R.O.C.
jslin@mail.ypu.edu.tw
[2]Department of Information Management, Yuanpei University, No.306, Yuanpei St., HsinChu, Taiwan, R.O.C.
liao@mail.ypu.edu.tw
[3]Department of General Education Center, Yuanpei University, No.306, Yuanpei St., HsinChu, Taiwan, R.O.C.
hsuch@mail.ypu.edu.tw

ABSTRACT

Cloud computing and cloud data storage have become important applications on the Internet. An important trend in cloud computing and cloud data storage is group collaboration since it is a great inducement for an entity to use a cloud service, especially for an international enterprise. In this paper we propose a cloud data storage scheme with some protocols to support group collaboration. A group of users can operate on a set of data collaboratively with dynamic data update supported. Every member of the group can access, update and verify the data independently. The verification can also be authorized to a third-party auditor for convenience.

Keywords

Cloud computing; Cloud data storage; Information security.

Academic Discipline And Sub-Disciplines

Information Management.

SUBJECT CLASSIFICATION

Information Techonology.

TYPE (METHOD/APPROACH)

Provable data possession, Proof of retrievability, Merkle hash tree, Aggregate group signature.

INTRODUCTION

Cloud computing (includes cloud data storage) allows users to obtain required services speedier than ever and easily to expand the services they need, with cheaper software acquisition and hardware maintenance costs. It also provides a flexible and convenient environment for users to access data and services. Furthermore, cloud computing allows people to create potential brand-new applications, such as automatic data backup and cross-regional group collaboration. Recently, group collaboration has become an important trend in cloud computing, and has been acknowledged as one of top 10 cloud computing trends for the decade [16]. In this paper, we propose a cloud data storage scheme with some protocols to support group collaboration. The proposed scheme possesses the following properties:

- **Proof of retrievability**: A Cloud storage service provider can provide a proof to a user to ensure that the data stored in the cloud servers are intact and retrievable.
- **Fully dynamic data updatable**: Users can make changes, including insertion, modification, deletion, and appending, to their data at their well at anytime.
- **Fully anonymous**: Non-group members cannot identify that a data change is made by which group member, and cannot distinguish that whether two data are modified by the same or different group members.
- **Fully traceable**: The group manager can find out that a data change is made by which group member, even though the change is generated by the collusion of multiple members.

RELATED WORKS

Early related works focused on peer-to-peer (P2P) network data storage problems. Lillibridge et al. proposed a scheme for P2P data backup by using (m+k,m)-erase codes to distribute file blocks to m + k peer hosts [14]. Filho et al. used RSA-based hash functions to verify data integrity, achieving undeceivable data authentication in P2P networks [10].

Ateniese et al. introduced the model of *provable data possession* (PDP) [1]. The main intention of PDP is to confirm the accuracy of data stored in untrusted storage servers. In the following research, Ateniese et al. used traditional symmetric key encryptions to construct their PDPs [3], providing more efficiency than the previous scheme, and supporting dynamic data file block appending and modification. Curtmola et al. extended the PDP model to multiple data replicas across distributed storage systems [6]. Their scheme can ensure the integrity of data without encoding each replica separately.

Juels et al. introduced the *proof of retrievability* (PoR) model to ensure the integrity of remote data [13]. Their scheme utilized error-correcting codes and pseudo-random dispersed checking blocks to ensure both possession and retrievability of data. Shacham et al. extended this PoR model with a random linear function, called *homomorphism authenticator*, and presented a PoR scheme without limitation on the number of verifications [17]. Bowers et al. generalized Juels's model and Shacham's model and presented an improved PoR scheme [4]. Latter, Bowers et al. expanded their scheme to a distributed architecture [5]. Dodis et al. linked PoR with the well studied topic *hardness amplification* in complexity theory and defined a purely information- theoretic notion of PoR codes [7]. Some improved PoR codes were also introduced. Recently, Wang et al. proposed a public auditing scheme for cloud data storage with zero knowledge privacy based on an aggregatable signature based broadcast (ASBB) encryption scheme [18]. Esiner1 et al. proposed a PoR scheme based on a data structure called FlexDPDP to impove the effeciency [9]. Han et al. proposed a PoR scheme with efficient aggregatable operations based on Maximum Rank Distance (MRD) codes [11].

Using erasure correction codes and homomorphism symbols, Wang et al. proposed a decentralized scheme which supports dynamic data modification, deletion, and appending (but no data insertion supported) [20]. Data errors can be detected and trying to find the location of the error blocks, recovering errors efficiently. In their follow-up study, Wang et al. utilized the *Merkle hash tree* (MHT) data structure to improve the PoR model, supporting both public verification and fully dynamic data update [19].

MHT is a widely employed authentication model, which is intended to effectively authenticate a set of data from an untrusted source with a small amount of trusted information [15]. A MHT is constructed as a binary tree, in which the leaves are the hashes of authentic data values. Figure 1 illustrates an authentication of a MHT with seven leaves. A verifier with the authentic root R requests for the data B_2 and B_5. Apart from providing the data, the prover also prepares some *auxiliary authentication information* (AAI) $A_2 = \langle G, D\rangle$ and $A_5 = \langle I, F\rangle$ for the verifier, where $G = h(B_1)$, $D = h(B_3)$, $I = h(B_4)$, $F = h(h(B_6) \,||\, h(B_7))$, || denotes concatenation operation, h is a public hash function. When received the data and the AAI from the prover, the verifier first computes $H' = h(B_2)$, $J' = h(B_5)$, $C' = h(G \,||\, H')$, $E' = h(I \,||\, J')$, $A' = h(C' \,||\, D)$, $B' = h(E' \,||\, F)$, and the root $R' = h(A' \,||\, B')$. Then the verifier compares R' and R. Accept if they are the same, reject otherwise.

Around the same time, Erway et al. extended the PDP model to a fully dynamically updatable scheme [8]. They utilized the *skip list* data structure to support dynamic data updates, in particular for data insertions. Wang et al. extended their scheme to a public-key encryption-based scheme which supports public verification and fully dynamic data update, and ensures no privacy leakage during public verifications [21]. To increase efficiency, this scheme also utilized aggregate signature to combine multiple verifications into one verification. Ateniese et al. proposed a general construction of public-key *homomorphic linear authenticator* (HLA) [2]. Any identification protocol can be transformed into a public-key HLA as long as the protocol satisfies appropriate conditions. Around the same time, Itani et al. presented some protocols to ensure the privacy of individual users in cloud data storage [12]. Recently, Zhou et al. proposed an attribute-based cloud data storage scheme for mobile devices [23]. However, they did not consider PDP or PoR.

ISSN 2277-3061

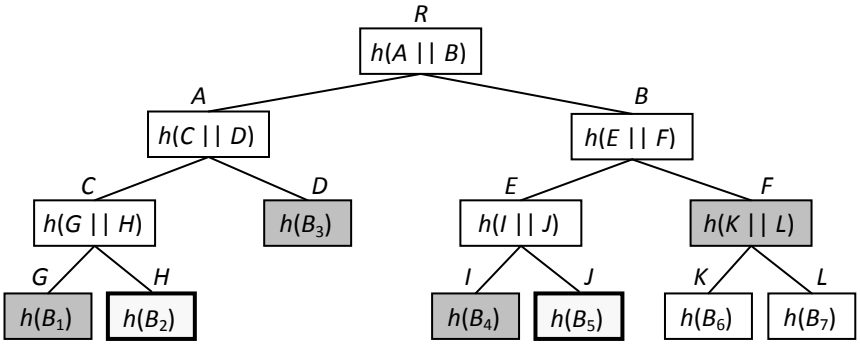

Figure 1. An example illustrates the authentication of a Merkle hash tree

ARCHITECTURE

A cloud data storage architecture for group collaboration is illustrated in Figure 2. In the architecture, messages between entities are transmitted by secured channels. The architecture consists of the following entities:

- User: A user is an individual who stores data in the cloud storage system. A user has full access right to his data and can verify the integrity and retrievability of the data at any time. A user may belong to one or more groups. In each group there is a group manager who is able to manage group members and trace a signature of the group to a member. All members of a group may work on a set of data on behalf of the group.
- Cloud Storage Service Provider (CSSP): A CSSP is an organization which possesses abundance of resources and expertise in order to construct and maintain a cloud data storage system.
- Third-Party Auditor (TPA): A TPA is an agency authorized by users or groups to verify the integrity and the retrievability of their data. In the cloud data storage architecture, TPA is an optional entity in the architecture.

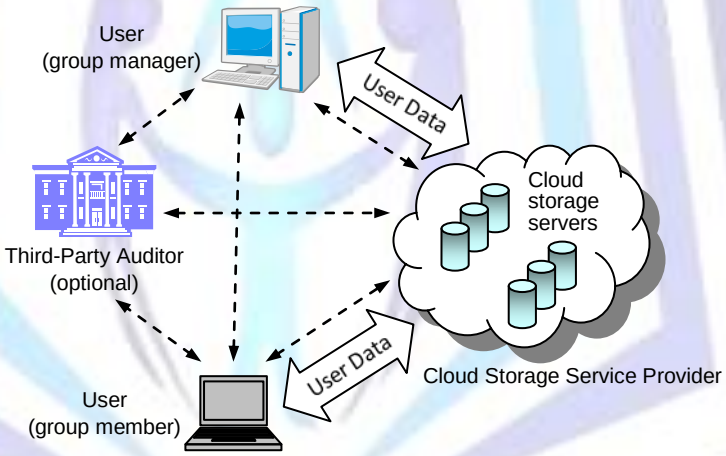

Figure 2. Cloud data storage architecture for group collaboration

A CSSP provides a vast amount of storage space that shared by all users. The storage space is usually constructed by multiple storage servers in a distributed manner. All storage servers work simultaneously and collaboratively. In order to ensure the accuracy of stored data, appropriate redundancies may be stored in the storage servers for the usage of error correction codes or erasure correction codes to prevent data loss due to accident or deliberate destruction. Users can access their private data through the interfaces provided by the CSSP and manipulate the data with appending, insertion, modification, and deletion operations according to their well. The redundancies in the storage servers must be adjusted immediately corresponding to the changes made by these operations. In order to let users fell relieved to store their data on the cloud storage system, there must be some ways to convince users that the data stored in the cloud are intact, confidential, and retrievable. For this purpose, we need an efficient method by which users can verify their data, and the verification will cause little computational load to the storage servers. Furthermore, the amount of the message transmitted between the users and the CSSP for the verification is as small as possible. When data are shared by a group of users, i.e. under group collaboration, each member can read, modify, and verify the shared data independently. The behavior of a group member should be concealed from outside of the group. That is, an entity outside a group cannot identify that a modification was made by which group member, or distinguish that two modifications were made by the same or different group members. However, the behavior of a group member should be traceable inside the group, i.e., given a modified data block, the group manager can disclose that which group member had made the modification.

A TPA is an institution trusted by users and has capability and expertise that users may not have. Users may not have sufficient capacity and resources, e.g. time, computing power and network bandwidth, to verify the accuracy of the data

stored in the cloud. In such a situation, a TPA can be authorized by users to verity their data immediately or periodically, and report results to the corresponding user.

PRPPOSED SCHEME AND PROTOCOLS

As described in the previous section, users and TPAs must have an efficient way to verify specific data stored in the cloud. In the proposed scheme, verifications are carried out by a challenge-response interaction. A user or a TPA can submit a request to the CSSP as a challenge. The CSSP then computes a set of values corresponding to the challenge and sends it back to the user or the TPA as a response. If the response coincides with the knowledge about the data, then it has proved that the data stored in the storage servers are intact and retrievable.

The complete scheme contains the following procedures: **Setup**, **GrpSetup**, **Join**, **ReqPermit**, **Enc**, **Dec**, **Sign**, **Verify**, **Open**, **Aggregate**, **AggVerify**, **SigGen**, **ReqProof**, **GenProof**, **ChkProof**, **ReqUpdate**, **ExecUpdate** and **ChkUpdate**. We adopt the methods in [19] and [22] to design these procedures. In the beginning, the scheme is initialized with **Setup** by a trusted party, generating public parameters. **GrpSetup** is called by a group manager to generate secret and public keys for a new group. A user calls **Join** to join a designed group, and calls **ReqPermit** to request one-time signing permits from the group manager when needed. When a user wants to store a file to the cloud storage, the file is first split into blocks of constant size, and then **SigGen** is called to generate metadata of each file block, then the file and its metadata are transferred to the server. In the execution of **SigGen**, it will call **Enc** to encrypt the file and call **Sign** to generate signatures of the file. We can use standard cryptographic primitives such as AES to implement **Enc**. When a user reads a file block, he first verifies it with **Verify**, and then decrypts it with **Dec**. When a user or a TPA wants to check the retrievability of a file, **ReqProof** is called to send a request to the CSSP. The CSSP uses **GenProof** to generate a proof according to the request and send it back to the requester, who then uses **ChkProof** to validate the proof. The proof includes an aggregated signature which is generated by **Aggregate** and can be verify by **AggVerify**. The purpose of using aggregated signature is to enhance verification efficiency and to save communication bandwidth. To update a file, a user sends an update request to the CSSP via **ReqUpdate**. The CSSP accomplishes the update with **ExecUpdate** and sends a proof for the update back to the user. The user then checks the proof by **ChkUpdate**. The manager of a group can use **Open** to find out which group member lastly updated a file block. The details of these procedures are described as follows.

- **Setup**(l): This procedure generates global parameters for the scheme according to the security parameter l, including a large prime power q of length l, an elliptic curve E over finite field F_q, a group order p of length l, two groups G_1 and G_2 of order p, a bilinear pairing $e : G_1 \times G_1 \rightarrow G_2$, a generator $P \in G_1$, and a map-to-point hash function $H_1 : \{0,1\}^* \rightarrow G_1$. The discrete logarithm problem (DLP) and the computational Diffie-Hellman problem (CDHP) should be hard in G_1, and the pairing e must satisfy the following properties:
  1. Bilinear: $e(aP_1, bP_2) = e(P_1, P_2)^{ab}$ for all $P_1, P_2 \in G_1$ and $a, b \in Z_p^*$.
  2. Non-degenerate: There exists $P_1, P_2 \in G_1$ such that $e(P_1, P_2) \neq 1$.
  3. Computable: There is an efficient algorithm to compute $e(P_1, P_2)$ for all $P_1, P_2 \in G_1$.
- **GrpSetup**(): A group manager uses this procedure to generate parameters for a new group. This procedure first randomly selects $s_A \xleftarrow{R} Z_p^*$ as group's private key and $s_E \xleftarrow{R} Z_p^*$ as secret key for encryption and decryption, and then computes the group public key $P_A = s_A P$. The group manager can then register this new group to the CSSP with P_A and *grpinfo* where *grpinfo* is the group description. The CSSP will record P_A and *grpinfo* in its database.
- **Join**(P_U, *usrinfo*, *grpinfo*): A user uses this procedure to join a group where P_U is user's public key and *usrinfo* is the user description. Before calling this procedure, the user selects $s_U \xleftarrow{R} Z_p^*$ as private key, and computes $P_U = s_U P$ in advance. If the manager agrees the request, he will compute a certificate $Cert \leftarrow s_A H_1(grpinfo \,||\, P_U \,||\, T)$ for the user, where T is a time period for this certificate. The manager will then record P_U, *usrinfo*, *Cert* and T to database and send back *Cert*, T, P_A and s_E to the user.
- **ReqPermit**(P_A, P_U, T, *Cert*, $\langle x_i P, x_i P_U \rangle_{1 < i < k}$): A user (group member) users this procedure to request k one-time signing permits from the group manager. The user first selects $x_1, x_2, \ldots, x_k \xleftarrow{R} Z_p^*$, and then computes $x_1 P$, $x_2 P$, . . . , $x_k P$ and $x_1 P_U$, $x_2 P_U$, . . . , $x_k P_U$. When the manager receives the request, he first exams the certificate by checking $e(Cert, P) \stackrel{?}{=} e(H_1(grpinfo \,||\, P_U \,||\, T), P_A)$, and tests the k pairs by checking $e(P, x_i P_U) \stackrel{?}{=} e(x_i P, P_U)$, $i = 1, \ldots, k$. The manager rejects the request if any of these tests fails. Otherwise the manager computes $S_i \leftarrow s_A H_1(grpinfo \,||\, x_i P_U \,||\, T)$, $i = 1, \ldots, k$, records to database these $x_i P_U$, $x_i P$ and S_i corresponding to P_U, and sends back S_i to the requester. A user may call this procedure whenever he needs a permit.
- **Enc**(s_E, B): Executed by a user to encrypt a data block B with the secret key s_E.
- **Dec**(s_E, C): Executed by a user to decrypt an encrypted data block C with the secret key s_E.
- **Sign**($x_i s_U$, $x_i P_U$, S_i, B, T): Executed by a user to sign a data block B with a signing permit S_i and the corresponding x_i s_U and $x_i P_U$. The procedure computes

$$S_B \leftarrow x_i s_U H_1(B), \quad S_G \leftarrow S_B + S_i .$$

  The signature is $\sigma \leftarrow (S_G, x_i P_U, T)$.

- **Verify**(σ, $H_1(B)$, P_A , *grpinfo*): Executed by an entity to verify a signature. Let $\sigma = (S_G , P_x , T)$. Accept if

$$e(S_G , P) = e(H_1(B), P_x) \cdot e(H_1(grpinfo \,||\, P_x \,||\, T), P_A),$$

and reject otherwise.

- **Open**(σ, B): Executed by the group manager to trace a data block B from its signature σ to a signer. Let $\sigma = (S_G , P_x , T)$. The group manager first verifies the signature. Then looks up the database to find the item which satisfies $x_i P_U = P_x$. This suffices to identify the user.

- **Aggregate**($\langle \sigma_j , H_1(B_j)\rangle_{i1 < j < im}$, P_A , *grpinfo*): The CSSP calls this procedure to combine multiple signatures into one aggregate signature, where $i_1, \ldots, i_m \in \{1, \ldots, n\}$ and $i_1 < \ldots < i_m$. Let $\sigma_j = (S_{Gj} , P_{xj} , T_j)$. The procedure first verifies each signature, then computes

$$S_\Sigma \leftarrow \sum_{j=1}^{m} S_{G_j} \ , \quad S_H \leftarrow \sum_{j=1}^{m} H_1(grpinfo \| P_{x_j} \| T_j) ,$$

and then returns (S_Σ, S_H , $\langle P_{xj} , H_1(B_j)\rangle_{i1 < j < im}$, P_A).

- **AggVerify**(S_Σ, S_H , $\langle P_{xj} , H_1(B_j)\rangle_{i1 < j < im}$, P_A): A user or a TPA uses this procedure to verify an aggregate signature. Accept if

$$e(S_\Sigma, P) = \prod_{j=1}^{m} e(H_1(B_j), P_{x_j}) \cdot e(S_H , P_A),$$

and reject otherwise.

- **SigGen**(s_E , $\langle x_i s_U , x_i P_U , S_i\rangle_{1 < i < n}$, F): A user utilizes this procedure to generate verification metadata for a file $F = \langle B_1 , B_2 , \ldots , B_n\rangle$. Here we assume that the file is pre-processed with Reed-Solomon code and divided into n blocks. The procedure first executes **Enc**(s_E , B_j) to encrypt each file block B_j to B_j', then obtains σ_j = **Sign**($x_i s_U$, $x_i P_U$, S_i , B_j') with a randomly chosen signing permit. Note that each signing permit should be used only once, and here we assume $k > n$. Finally, the procedure constructs the MHT from these B_j' and then computes the signature σ_{RF} of the MHT root R_F.

  When this procedure is finished, the user sends CSSP the file description *fileinfo*, the encrypted file $F' = \langle B_i'\rangle$, the set of signatures $\Phi = \langle \sigma_i\rangle$, and the signed MHT root σ_{RF} corresponding to the file. The user doesn't need to keep the file and its signature, thus he can delete them from his local storage. Note that one file needs only one execution of this procedure.

- **ReqProof**(*fileinfo*, *chal*): A user or a TPA sends a request to the CSSP for the verification of a file F by calling this procedure. The parameter *chal* contains a set of block indices $I = (i_1, \ldots, i_m)$ which indicates the CSSP to generate a proof for this verification, where i_j is randomly selected from $\{1, \ldots, n\}$ and we assume $i_1 < \ldots < i_m$.

- **GenProof**(F', Φ, P_A , *grpinfo*, *chal*): The CSSP generates a proof V for the verification of a file F via this procedure. The procedure first retrieves the file blocks and their signatures from F' and Φ according to the set of block indices $I = (i_1, \ldots, i_m)$ contained in *chal*, and calls **Aggregate**($\langle \sigma_j , H_1(B_j)\rangle_{i1 < j < im}$, P_A , *grpinfo*) to obtain an aggregated signature (S_Σ, S_H , $\langle P_{xj} , H_1(B_j)\rangle_{i1 < j < im}$, P_A). Then it computes some auxiliary authentication information (AAI) $\langle A_j\rangle_{i1 < j < im}$ based on I and the MHT of the file where A_j is the set of node siblings on the path from the leaf $H_1(B_j')$ to the root R_F. Finally, the procedure returns $V = (S_\Sigma, S_H , \langle P_{xj} , H_1(B_j'), A_j\rangle_{i1 < j < im} , P_A , grpinfo, \sigma_{RF})$ where σ_{RF} is the signature of R_F.

  After the procedure is complete, the CSSP sends V to the verifier for verification.

- **ChkProof**(V): A user or a TPA uses this procedure to validate the proof V. The procedure first verifies the aggregated signature with **AggVerify**. Then it obtains the MHT root R_F' with $\langle H_1(B_j'), A_j\rangle_{i1 < j < im}$, and checks whether σ_{RF} is a valid signature of R_F'. The procedure accepts the proof if all these checks are successful, otherwise rejects the proof.

- **ReqUpdate**(*fileinfo*, *inst*): A user updates a file with this procedure. The parameter *inst* consists of an instruction (modify, insert, or delete), a block index, a new block and its signature (optional) for the update. Data appending can be accomplished by insertion.

- **ExecUpdate**(F', Φ, *inst*): The CSSP utilizes this procedure to accomplish the update of the file F according to the parameter *inst*. The operation includes storing the new file block and its signature (modify and insert) or removing the designate block and its signature (delete), and then adjusting the MHT of the file accordingly. We adapt the method in [19] to perform this adjustment. After the update is complete, the procedure returns a proof $V = (H_1(B_i'), A_i , \sigma_{RF}' , R_F)$ which will then passes to the requester, where i is the index of the updated block, B_i' is the original block, σ_{RF}' is the signature of the MHT root before update, and R_F is the MHT root after update.

- **ChkUpdate**(*inst*, V): A user uses this procedure to check whether the CSSP has updated a file correctly according to *inst* or not. The procedure first obtains the original MHT root R_F' with the AAI $H_1(B_i')$ and A_i , and then checks whether σ_{RF}' is a valid signature of R_F'. The objective of this check is to authenticate the AAI. If the check fails, the procedure terminates and returns *FALSE*. Otherwise, it proceeds to check whether the CSSP has updated the file as required or not. It computes the updated MHT root with the new block (included in *inst*) and compares the new root with R_F. If they are the same, then the procedure returns *TRUE*, or returns *FALSE* otherwise. The user will sign the new root and send its signature to the CSSP when this procedure returns *TRUE*.

Theorem 1 The aggregate signature in the proposed scheme is well defined.

Proof: For a legal signature $\sigma = (S_G , x_i P_U , T)$ of a data block B, assuming the related group public key is P_A, then

$$e(S_G , P) = e(S_B + S_i , P)$$

$$= e(x_i x_U H_1(B), P) \cdot e(s_A H_1(grpinfo \,||\, x_i P_U \,||\, T) , P)$$

$$= e(H_1(B), P_x) \cdot e(H_1(grpinfo \,||\, P_x \,||\, T), P_A).$$

So the **Verify** procedure can successfully verify the signature. Furthermore, for an aggregate signature $(S_\Sigma , S_H , \langle P_{xj} , H_1(B_j)\rangle_{i1 < j < im} , P_A)$.

$$e(S_\Sigma, P) = e(\sum_{j=1}^{m} S_{G_j} , P)$$

$$= e(\sum_{j=1}^{m} S_{B_j} , P) \cdot e(\sum_{j=1}^{m} S_{i_j} , P)$$

$$= e(\sum_{j=1}^{m} s_{U_j} x_{i_j} H_1(B_j) , P) \cdot e(\sum_{j=1}^{m} s_A H_1(grpinfo \| P_{x_j} \| T_j) , P)$$

$$= \prod_{j=1}^{m} e(H_1(B_j), P_{x_j}) \cdot e(S_H , P_A).$$

Thus the **AggVerify** procedure can also successfully verify the aggregate signature. □

SECURITY ANALYSIS

The security of the proposed scheme is mainly based on the hardness of the elliptic curve discrete logarithm problem and some related problems on elliptic curves.

Elliptic curve discrete logarithm problem (ECDLP): Given two points P and Q in a rational point group on an elliptic curve, find an integer k such that $kP = Q$. k is called the discrete logarithm of Q to the base P.

Computation Diffie-Hellman problem (CDHP): Let G_1 be a cyclic rational point group of order p on an elliptic curve, P be a generator of G_1. Given aP and bP, $a, b \in Z_p^*$, compute abP.

Decisional Diffie-Hellman problem (DDHP): Let G_1 be a cyclic rational point group of order p on an elliptic curve, P be a generator of G_1. Given aP, bP and cP, $a, b, c \in Z_p^*$, decide whether $c \equiv ab \bmod p$ is satisfied or not.

If the DDHP in G_1 can be solved in polynomial time, but the ECDLP and CDHP cannot be solved in polynomial time, G_1 is called a gap Diffie-Hellman group. Our scheme is based on such a group.

The proposed scheme may suffer from several types of attack. The first type of attack is to obtain the private key of a group or a group member. The second type of attack is to forge the data stored in cloud storage. The attacker may be an external attacker, the CSSP itself, an individual within the group, or some colluded members within the group. The third type of attack is to steal the data stored in cloud storage. The fourth type of attack is to destroy the integrity of the data stored in cloud storage.

Type I attack: obtain a private key

1. **Obtain the private key of a group**. There are two ways to obtain the private key of a group. The first way is to compute the private key s_A from the group public key $P_A = s_A P$. However, this is equivalent to solve the eclipse curve discrete logarithm problem. The second way is that a member of the group can try to compute s_A from his certificate $Cert = s_A H(grpinfo \,||\, P_U \,||\, T)$ or from an one-time signing permit $S_i = s_A H(grpinfo \,||\, x_i P_U \,||\, T)$. However, this is also equivalent to solve the eclipse curve discrete logarithm problem.
2. **Obtain the private key of a group member**. There are two ways to obtain the private key of a group member. The first way is to compute the private key s_U from his public key $P_U = s_U P$. However, this is equivalent to solve the eclipse curve discrete logarithm problem. Another possible way is to solve the private key from a signature $\sigma_i = (S_G , x_i P_U , T)$ signed by the member where $S_G = x_i s_U H(B) + s_A H(grpinfo \,||\, x_i P_U \,||\, T)$. Since every signature uses a distinct x_i, solving this equation is equivalent to solve the eclipse curve discrete logarithm problem. It is no exception even the group manager knows s_A.

Type II attack: forge the data stored in cloud storage

An adversary may fake a data block, and then forge the signature of this data block. To successfully fake a data block, a legal signature must be forged. Suppose the signature of a faked data block is $\sigma_i = (S_G , x_i P_U , T)$. this signature must pass the following examination

$$e(S_G , P) = e(H(B), x_i P_U) \cdot e(H(grpinfo \,||\, x_i P_U \,||\, T), P_A) \quad (1)$$

Assuming an adversary randomly chooses an x_i, an s_{Ui}, and sets up a time period T. The adversary then computes $P_U = s_U P$, $x_i P_U$, and $H(grpinfo \,||\, x_i P_U \,||\, T)$. Since the adversary does not know s_A, he cannot obtain a valid S_G which satisfies (1). However, if the adversary is the group manager, he can forge a signature that does not belong to any member. Nevertheless, because only the group manager has this ability, in this case, we know that the group manager is the adversary.

Type III attack: steal the data stored in cloud storage

An adversary may steal the data stored in cloud storage. But because he does not know the correct decryption key, the confidentiality of the data is still preserved.

Type IV attack: destroy the integrity of the data stored in cloud storage

An adversary may deliberately undermine the integrity of the data stored in cloud storage. Even the CSSP itself may also damage the integrity of user's data intentionally or unintentionally. No matter in which situation, the constructed MHT root node will not match the original root node signed by the user. Therefore, when a user or a TPA verifies the data, he can find out the error. When an error is found, the user can further examine each data block to find out which data block is destroyed, and then corrects the error.

DISCUSSIONS AND CONCLUSIONS

Cloud computing and cloud data storage have become important applications on the Internet. Technology giants such as Microsoft, Amazon, Google, IBM, Cisco, and Dell have invested in developing cloud computing and data storage technologies and services. In such a development, group collaboration is an important trend since it is a great inducement for an entity to use a cloud service, especially for an international enterprise. In this paper we propose a cloud data storage scheme with some protocols to support group collaboration. A group of users can operate on a set of data collaboratively with dynamic data appending, insertion, modification, and deletion operations. Every member of the group can access, update and verify the data independently. The verification can also be authorized to a TPA for convenience. The TPA cannot learn information about the data and the user. However, the group manager can find out which member lastly updated a file block.

The security of our scheme is based on the hardness of the ECDLP and the CDHP on finite fields. In order to provide sufficient security (approximately the same as the standard 1024-bit RSA signature), we can use elliptic curves over finite field F_q with embedding degree 6 where q is approximately 170 bits long. G_1 of prime order p is a subgroup of $E(F_q)$ where p is also of length approximately 170 bits. Therefore, the ECDLP in G_1 is as hard as the DLP in finite field of length approximately 1020 bits. For a data block, the signature is of size about 350 bits, or 44 bytes. For a 1GB file with data block of size 8KB, there are totally 131,073 signatures (plus the one for the MHT root), resulting about 5.5MB overhead to be stored in the cloud storage for this file. Signing a file block needs one hash operation, one elliptic curve multiplicative operation, and one elliptic curve additive operation. Verifying a file block costs two hash operations, three pairing operations, and one finite field multiplicative operation. The size of an aggregate signature is linear to the number of data blocks requested. To aggregate m signatures it takes m hash operations and totally $m - 1$ elliptic curve additive operations. Verifying an aggregate signature with m signatures aggregated spends $m + 2$ pairing operations and $2m - 1$ finite field multiplicative operations, and it brings around $132 + 88m$ bytes communication throughput for this verification. Overall speaking, our scheme is pretty efficient.

REFERENCES

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. 2007. Provable Data Possession at Untrusted Stores. Proc. of CCS '07, pp. 598–609, 2007.

[2] G. Ateniese, S. Kamara, and J. Katz. 2009. Proofs of Storage from Homomorphic Identification Protocols. ASIACRYPT 2009, pp. 319-333.

[3] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik. 2008. Scalable and Efficient Provable Data Possession. Proc. of SecureComm '08, pp. 1-10, 2008.

[4] K. D. Bowers, A. Juels, and A. Oprea. 2008. Proofs of Retrievability: Theory and Implementation. Cryptology ePrint Archive, available at http://eprint.iacr.org/2008/175.

[5] K. D. Bowers, A. Juels, and A. Oprea. 2008. HAIL: A High-Availability and Integrity Layer for Cloud Storage. Cryptology ePrint Archive, available at http://eprint.iacr.org/2008/489.

[6] R. Curtmola, O. Khan, R. Burns, and G. Ateniese. 2008. MR-PDP: Multiple-Replica Provable Data Possession. Proc. of ICDCS '08, pp. 411–420, 2008.

[7] Y. Dodis, S. Vadhan, and D. Wichs. 2009. Proofs of retrievability via hardness amplification. Theory of Cryptography Conference, LNCS 5444, pp. 109-127, Springer-Verlag, 2009.

[8] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia. 2008. Dynamic provable data possession. Cryptology ePrint Archive, available at http://eprint.iacr.org/2008/432.

[9] E. Esiner1, A. Kachkeev1, S. Braunfeld, A. Küpçü, and Ö. Özkasap. 2013. FlexDPDP: FlexList-based Optimized Dynamic Provable Data Possession. Cryptology ePrint Archive, available at http://eprint.iacr.org/2013/645.

[10] D. L. G. Filho and P. S. L. M. Barret. 2006. Demonstrating Data Possession and Uncheatable Data Transfer. Cryptology ePrint Archive, available at http://eprint.iacr.org/2006/150.

[11] S. Han, S. Liu, K. Chen, and D. Gu. 2013. Proofs of Data Possession and Retrievability Based on MRD Codes. Cryptology ePrint Archive, available at http://eprint.iacr.org/2013/789.

[12] W. Itani, A. Kayssi, A. Chehab. 2009. Privacy as a service: privacy-aware data storage and processing in cloud computing architectures. Proceedings of the 2009 International Conference on Dependable, Autonomic and Secure Computing (DASC 2009), pp. 711-16, 2009.

[13] A. Juels and J. Burton S. Kaliski. 2007. PORs: Proofs of Retrievability for Large Files. Proc. of CCS '07, pp. 584–597, 2007.

[14] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard. 2003. A Cooperative Internet Backup Scheme. Proc. of the 2003 USENIX Annual Technical Conference (General Track), pp. 29–41, 2003.

[15] R. C. Merkle. 1980. Protocols for public key cryptosystems. Proc. of IEEE Symposium on Security and Privacy'80, pp. 122–133, 1980.

[16] S. Mitra and S. Shalini. 2011. Top 10 Cloud Computing Trends for the Decade. available at http://anatango.wordpress.com/2011/08/17/top-10-cloud- computing- trends-for-the-decade.

[17] H. Shacham and B. Waters. 2008. Compact Proofs of Retrievability. Proc. of Asiacrypt '08, Dec. 2008.

[18] S.-H. Wang, S.-Q. Chang, D.-W. Chen, and Z.-W. Wang. 2012. Public Auditing for Ensuring Cloud Data Storage Security With Zero Knowledge Privacy. Cryptology ePrint Archive, available at http://eprint.iacr.org/2012/365.

[19] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. 2009. Enabling public verifiability and data dynamics for storage security in cloud computing. In Proc. of ESORICS'09, Saint Malo, France, Sep. 2009.

[20] C. Wang, Q. Wang, K. Ren, and W. Lou. 2009. Ensuring data storage security in cloud computing. Proc. of IWQoS'09, Charleston, South Carolina, USA, 2009.

[21] C. Wang, Q. Wang, K. Ren and W. Lou, "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing," INFOCOM 2010, pp. 525-533.

[22] D. Yao and R. Tamassia. 2006. Cascaded Authorization with Anonymous-Signer Aggregate Signatures. In Information Assurance Workshop, 2006 IEEE, pp. 84-91, June 2006.

[23] Z. Zhou and D. Huang.2011. Efficient and Secure Data Storage Operations for Mobile Cloud Computing, Cryptology ePrint Archive, available at http://eprint.iacr.org/2011/185.

Author' biography with Photo

Jyh-Shyan Lin was born in Kaohsiung, Taiwan, 1968. He received the M.S. degrees in Computer Science and Information Engineering from the National Chung Cheng University, Chiayi, Taiwan, in 1999. He received his Ph.D. degree in Computer Science and Information Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2007. He is currently an Assistant Professor of the Department of Information Management, Yuanpei University, Hsinchu, Taiwan. His research is mainly in the field of cryptography, coding theory, algorithms, cloud computing, and bioinformatics.

Kuo-Hsiung Liao is an assistant professor of the Department of Information Management at the Yuanpei University, HsingChu, Taiwan,R.O.C. He received the M.S. degree in computer science from the New York Institute of Technology, in1992. He has published papers in the fields of e-government, and medical information management. His research interests include neural networks, computer security, programming language, artificial intelligent, medical information management and pattern recognition.

Chao-Hsing Hsu is a lecture of the General Education Center at the Yuanpei University, HsingChu, Taiwan,R.O.C. He received the M.S. degree in Mathematics from the Fu Jen Catholic University, in1987. He has published papers in the fields of applied mathematics, biostatistics, and medical information management. His research interests include calculus, applied mathematics, biostatistics.