Strong Key-Exposure Resilient Auditing for Secure Cloud Storage

Jia Yu, and Huaqun Wang

Abstract—Key exposure is one serious security problem for cloud storage auditing. In order to deal with this problem, cloud storage auditing scheme with key-exposure resilience has been proposed. However, in such a scheme, the malicious cloud might still forge valid authenticators later than the key-exposure time period if it obtains the current secret key of data owner. In this paper, we innovatively propose a paradigm named strong key-exposure resilient auditing for secure cloud storage, in which the security of cloud storage auditing not only earlier than but also later than the key exposure can be preserved. We formalize the definition and the security model of this new kind of cloud storage auditing and design a concrete scheme. In our proposed scheme, the key exposure in one time period doesn’t affect the security of cloud storage auditing in other time periods. The rigorous security proof and the experimental results demonstrate that our proposed scheme achieves desirable security and efficiency.

Index Terms—Cloud computing; cloud storage auditing; data integrity; key exposure

I. INTRODUCTION

NOWADAYS, cloud storage is becoming one of the most attractive choices for individuals and enterprises to store their large scale of data. It can avoid committing large capital of users for purchasing and managing hardware and software. Although the benefits of cloud storage are tremendous, security concerns become significant challenges for cloud storage. One major concern on cloud storage security is about the integrity of the data stored in cloud. Because clients lose the control of their data stored in cloud and cloud loss happen in cloud storage, it is natural for clients to doubt whether their data are correctly stored in cloud or not. Cloud storage auditing, as one effective security technique, is proposed to ensure the integrity of the data stored in cloud.

Many cloud storage auditing schemes have been proposed up to now [1–13]. These schemes consider several different aspects of cloud storage auditing such as the data dynamic update [5, 6], the privacy protection of user’s data [7, 8], the data sharing among multiple clients [9, 10] and the multicopies of cloud data [11, 12]. Key-exposure resilience, as another important aspect, has been proposed recently [13]. Indeed, the secret key might be exposed due to the weak security sense and/or the low security settings of the client. Once a malicious cloud gets the client’s secret key for cloud storage auditing, it can hide the data loss incidents by forging the authenticators of fake data. As the same reason, it even can discard the client’s rarely accessed data for saving the storage space without being found out by cloud storage auditing. In [13], a key update technique based on binary tree structure is used to protect the security of authenticators generated in time periods earlier than the key exposure. As a result, the cloud storage auditing scheme in [13], to some extent, can deal with the key exposure problem.

However, in some cases, the key exposure problem is not fully solved in the scheme [13] due to the following reason. When the key exposure happens, it often cannot be found out at once. The key exposure might be difficult to be found out because the attacker might stop intrusion at once when it gets the client’s secret key. So it is common that there is a long time span crossing multiple time periods between key exposure and its detection. The key exposure might be detected only when the user finds the valid authenticators are not generated by himself. At that time, the user has to revoke the old pair of public key and secret key, and regenerate a new pair. We give an example to show this problem in Fig. 1. Suppose the hacker has appropriated the client’s secret key during session $t_{e}$, but the key exposure has not been detected at that time. The attacker can update the exposed secret key, as same as the client does, to generate the secret keys for time periods $t_{e}, ..., t_{d}$ until key exposure is found out during time period $t_{d}$. This means that the malicious cloud with this hacker can modify even delete the client’s data uploaded during time periods $t_{e}, ..., t_{d}$ without worrying about being found out. It can generate the authenticators for fake data to pass the cloud storage auditing using the updated secret keys. It is a natural problem of how to protect the security of the cloud storage auditing during the time periods not only earlier than but also later than the key exposure.

A. Contribution

The contribution of this paper can be summarized as follows:

1) We investigate how to preserve the security of cloud storage auditing scheme in any time period other than the key-exposure time period when the key exposure happens. We propose a paradigm named strong key-exposure resilient auditing as a practical solution for this problem in this paper.

2) We design a concrete strong key-exposure resilient auditing scheme for secure cloud storage. A novel and efficient key update technique is used in the designed scheme. In our detailed construction, the Third Party Auditor (TPA) generates...
B. Related work

In recent years, a lot of studies on checking the integrity of the data stored on untrusted servers have been done. The notion of Provable Data Possession (PDP) was firstly proposed by Ateniese et al. [14] for ensuring data possession on untrusted servers. This scheme checked the integrity of outsourced data by the techniques of random sample and homomorphic linear authenticators. Juels and Kaliski [15] explored the model named as Proof of Retrievalability (PoR) which can ensure both possession and retrievability of the files on untrusted servers. They used the techniques of error-correcting codes and spot-checking to construct the PoR scheme. Shacham and Waters [16] provided an improved PoR model with stateless verification. They proposed a private verification scheme based on pseudorandom functions and a public verification scheme based on BLS signature scheme. In [17], Dodis et al. studied on different variants of the existed PoR work. In [7], Wang et al. integrated the HLA with random masking technique to make the auditor unable to infer the original data from auditing process.

The PDP supporting for data dynamic operations was firstly researched in [18]. Wang et al. [5] proposed another cloud storage auditing scheme that supported data dynamics by utilizing the BLS-based HLA and Merkle Hash Tree. Erway et al. [19] proposed a PDP scheme to support data dynamics using a skip list-based structure. Zhu et al. [20] proposed a cooperative provable data possession scheme. Yang and Jia [8] considered the dynamic operation and privacy-preserving property in cloud storage auditing scheme. Cash et al. [21] proposed a dynamic PoR scheme using oblivious ram technique. Some other important researches about dynamic cloud storage auditing [22, 23] have been done. The problem of user reversion in shared cloud data auditing was considered in [10]. Guan et al. [24] proposed a cloud storage auditing scheme for low-power clients based on indistinguishability obfuscation. Identity-based cloud storage auditing schemes were proposed to simplify key management process in [25, 26]. Multiple-replica cloud storage auditing schemes were proposed in [11, 12]. Identity privacy and identity traceability for shared cloud storage were studied in [27] and [28].

Recently, key exposure problem and its verifiable outsourcing of key updates for cloud storage auditing have been considered in [13] and [29], respectively. In [13], the client’s secret keys are updated in different time periods. The key exposure cannot affect the security of authenticators generated before the key-exposure time period. However, as we have analyzed, it cannot fully solve the key exposure problem in some cases, i.e., the security of authenticators generated later than the key-exposure time period is still unable to preserve. Therefore, the contributions of this paper can be viewed as the further research on the key exposure problem in cloud storage auditing.

C. Organization

The rest paper is organized as follows: In Section 2, we introduce system model, definition, security model and preliminaries. Then, we give a detailed description of the proposed scheme in Section 3. The security analysis and the efficiency evaluation are given in Section 4. At last, we conclude the paper in Section 5.

II. FORMULATION AND PRELIMINARIES

A. System model

The system comprises three parties: the cloud, the client and the third party auditor (TPA). The cloud offers storage services to the client. The client uploads his files along with the corresponding authenticators to the cloud, and then deletes these data from his storage space. The client can retrieve them from the cloud when he needs them. The TPA is a powerful party and is in charge of two important tasks. The first is to provide auditing service, i.e., periodically check the integrity of the files stored in cloud for the client. The second is to help the client update his secret keys by providing update messages to the client in different time periods. As same as
most of public integrity auditing schemes, the TPA is honest for integrity auditing on behalf of cloud users. However, it is not fully trustworthy for key update in our system model. The system model is shown in Fig. 2. Assume that one file $F$ stored in cloud is divided into $n$ blocks $m_i (i = 1, ..., n)$. Different from the system model in [13], the lifetime of file $F$ in our system model does not need to be fixed initially. It means the total number of time periods in unbounded, which is more close to the reality.

B. Definition and Security Model

Definition 1: A strong key-exposure resilient auditing scheme for secure cloud storage consists of six algorithms such that:

1) System setup algorithm $SysSetup$: This algorithm is run by the client. It takes as input a security parameter $k$, and generates a system public key $PK$, the TPA's secret key $SK_{TPA}$ and the client's private key $SK_c$.

2) Update message generation algorithm $UMGen$: This algorithm is run by the TPA at the beginning of each time period. It takes as input the public key $PK$, the current time period $t$ and the TPA's secret key $SK_{TPA}$, and generates an update message $\delta_t$.

3) Client key update algorithm $CKeyUpdate$: This algorithm is run by the client at the beginning of each time period. It takes as input the public key $PK$, the current time period $t$, the update message $\delta_t$ and the client's private key $SK_c$, and generates the signing secret key $SK_t$ for time period $t$.

4) Authenticator generation algorithm $AuthGen$: This algorithm is run by the client. It takes as input the public key $PK$, the current time period $t$, the client's signing secret key $SK_t$ and a file $F$, and generates a set of authenticators $\Phi$ for $F$ in time period $t$.

5) Proof generation algorithm $ProofGen$: This algorithm is run by the cloud. It takes as input the public key $PK$, a time period $t$, a challenge $Chal$, a file $F$ and a set of authenticators $\Phi$, and generates a proof $P$ that is used to prove the cloud stores $F$ correctly. In this algorithm, the $(t, Chal)$ pair is issued by the TPA, and then used by the cloud as input.

6) Proof verifying algorithm $ProofVerify$: This algorithm is run by the TPA. It takes as input the public key $PK$, a time period $t$, a challenge $Chal$ and a proof $P$, and returns “true” if the verification passed; or “false”, otherwise.

Note that the client’s secret information includes two parts. One is the signing secret key $SK_t$, which is used to generate authenticators in time period $t$. The other is the private key $SK_c$, which is used to generate the signing key $SK_t$ in time period $t$. In order to formalize the security, we define a game between an adversary $A$ and a challenger $C$ to show how the adversary attacks the security of a strong key-exposure resilient auditing scheme. In this game, we consider the most powerful adversary who can query the secret keys of the client in all except one unexposed time period. This game comprises the following four phases.

1) Setup phase. Initially set time period $t=0$. $C$ runs $SysSetup$ algorithm to generate the public key $PK$, the TPA's secret key $SK_{TPA}$ and the client's secret key $SK_c$. $C$ sends $PK$ to $A$ and holds $SK_c$ himself at this phase.

2) Query phase. $C$ runs key update algorithm to generate the client's signing key $SK_t$ in time period $t$.
   I. Authenticator queries. $A$ adaptively selects a series of blocks $m_1, ..., m_n$, and submits them to $C$ to query the corresponding authenticators in time period $t$. The challenger computes and returns the corresponding authenticators for $m_i (i = 1, ..., n)$ in this time period.
   II. Secret key queries. $A$ can select to query the secret key of the client in time period $t$. If he does, $C$ sends the client's private key $SK_c$ and signing key $SK_t$ in time period $t$ to $A$.
   At the end of each time period, $A$ can select to stay in this phase or go to the Challenge phase.

3) Challenge phase. $C$ selects a time period $t^*$ in which $A$ does not make the secret key queries. $C$ sends $A$ a challenge $Chal=\{i, v_i\}_{i \in I}$ (where $I = \{s_1, s_2, ..., s_c\}$, $1 \leq s_l \leq n$, $1 \leq l \leq c$, $1 \leq c \leq n$). It also requests $A$ to provide a proof of possession $P$ for file $F = (m_1, ..., m_n)$ under $Chal$ for the blocks $m_{s_1}, ..., m_{s_c}$ in time period $t^*$.

4) Forgery phase. $A$ outputs a proof of possession $P$ for the blocks indicated by $Chal$ in the time period $t^*$, and returns $P$. If $ProofVerify(PK, t^*, Chal, P)="true"$, then $A$ wins in above game.

The above security model captures that an adversary cannot provide a valid proof for a time period in which the secret key is not exposed without owning all the blocks corresponding to a given challenge, if it cannot guess all the missing blocks. In each time period, the adversary can query the authenticators of all the blocks. The adversary can also query the secret keys in all except the challenged time period. The goal of the adversary is to construct a valid proof of possession $P$ for the blocks indicated by $Chal$ in the time period $t^*$. The following definition shows that there exists a knowledge extractor that can extract the challenged file blocks whenever the adversary can produce valid proof of possession $P$ in the time period $t^*$.

Definition 2: We say a cloud storage auditing scheme...
is strong key-exposure resilient if the following condition satisfies: whenever an adversary in above described game can cause the challenger to accept its proof with non-negligible probability, there is a knowledge extractor that can extract all the challenged file blocks except possibly with negligible probability.

The following definition describes the detectability for the auditing scheme, which shows the cloud should maintain the blocks that are not challenged with high probability.

**Definition 3**: We say a cloud storage auditing scheme is \((\rho, \delta)\) detectable if, given a fraction \(\rho\) of corrupted blocks, the probability that the corrupted blocks are detected is at least \(\delta\).

In above system model, the TPA generates an update message to help the client update his secret key in each time period. Another security requirement is that the TPA is not able to forge any authenticator through his secret key and the generated update messages. We give the following security definition to capture this requirement.

**Definition 4**: The TPA cannot forge the authenticator for any block through his secret key and the update messages he generates.

The TPA might incorrectly execute the update key generation algorithm. That is, the TPA might provides the client wrong update key for some reason. In this case, the client should be able to verify the correctness of the update key provided by the TPA.

**Definition 5**: We say a strong key-exposure resilient auditing for secure cloud storage is verifiable if the TPA provides the wrong update key in **UMGen** algorithm, he cannot pass the verification of our proposed scheme.

**C. Preliminaries**

1. **Bilinear map**

   There are two multiplicative cyclic groups \(G_1\) and \(G_2\), which have the same prime order \(q\). We say a map \(\hat{\cdot} : G_1 \times G_1 \rightarrow G_2\) is a bilinear map if it satisfies:
   1. **Bilinearity**: For all \(g_1, g_2 \in G_1\) and \(a, b \in Z_q^*\), \(\hat{\cdot}(g_1^a, g_2^b) = \hat{\cdot}(g_1, g_2)^{ab}\).
   2. **Non-degeneracy**: \(\hat{\cdot}(g_1, g_2) \neq 1\), where \(g_1, g_2\) are the generators of \(G_1\).
   3. **Computability**: There is an efficient algorithm to compute \(\hat{\cdot}(g_1, g_2)\) for \(g_1, g_2 \in G_1\).

2. **CDH problem**

   Given \((g, g^a, g^b)\), where \(g\) is a generator of multiplicative group \(G_1\) with order \(q\), and \(a, b \in Z_q^*\), compute \(g^{ab}\).

**III. OUR PROPOSED SCHEME**

**A. Technique Explanation**

One big challenge of designing such a scheme is that the signing secret keys change in different time periods while the public key is unchanged in all of time periods. We design a new key update technique that is different from that of [13]. In order to achieve the strong key-exposure resilience, we make the signing secret key in each time period be a multiplication of two parts. Each part is the power of \(H_1(t)\), where \(H_1\) is a hash function and \(t\) is the current time period. One part is the update message generated by the TPA, which is computed through the secret key of the TPA and the current time period. The other part is computed from the secret key of the client and the current time period. The signing secret key in any time period must be jointly generated by the client and the TPA. This technique can support both the provable security and the efficient key update. As a result, if the attacker intrudes the client in one time period, he cannot obtain the client’s signing secret keys in other time periods without the secret key of the TPA. The designed authenticator can support the structure of signing secret keys and the property of blockless verifiability. Because the time period as an important factor is integrated into the computation of authenticators, the authenticators of the same file blocks generated in different time periods are different. The **ProofVerify** algorithm can verify whether the proof corresponding to the declared time period is indeed valid or not.

**B. Description of Our Scheme**

Let \(\hat{\cdot} : G_1 \times G_1 \rightarrow G_2\) be a bilinear map, where \(G_1\) and \(G_2\) are two multiplicative groups with order \(q\). Let \(g\) and \(u\) be two generators of group \(G_1\), and \(H_1 : \{0, 1\}^* \rightarrow G_1, H_2 : \{0, 1\}^* \times G_1 \rightarrow G_1\) be two cryptographic hash functions. Generally, there is a digital signature \(SSig\) that is used to ensure the integrity of the file identifier **name** in previous cloud storage auditing schemes [5, 7, 13]. In this paper, we also use the same digital signature \(SSig\) to ensure the integrity of the file identifier **name** and the time period \(t\). We assume \((spk, ssk)\) is a pair of public key and secret key corresponding to signature \(SSig\), the client has held the secret key \(ssk\), and the public key \(spk\) has been published. Such an assumption can simplify our scheme description thereafter.

Similar to previous cloud storage auditing schemes [2, 4, 5, 7, 8, 13], the client firstly divides one file \(F\) stored on the cloud into a set of \(n\) ordered blocks \(m_1, m_2, ..., m_n\), where \(m_i \in Z_q^*\) in our scheme. In time period \(t\), the signing secret key of the client is \(SK_i \in G_1\). The authenticator for each block \(m_i\) in time period \(t\) is generated as follows. The client selects a random \(r \in Z_q^*\) and computes \(R = g^r\). He computes the authenticator for each block \(m_i\) in time period \(t\) as \(\sigma_i = H_2(t || name, R^r \cdot w^{m_i} \cdot SK_i\), where **name** is the name of the file \(F\).**

1) **SysSetup algorithm**

   a) The client randomly selects \(SK_c \in Z_q^*\) as his private key. The client’s public key is \(PK_c = g^{SK_c}\).
   b) The client randomly selects \(SK_{TPA} \in Z_q^*\), and sends it to the TPA as the TPA’s secret key. The TPA’s public key is \(PK_{TPA} = g^{SK_{TPA}}\).
   c) Set the system public key \(PK = (g, u, PK_{TPA}, PK_c, spk)\).

2) **UMGen algorithm**

   a) At the beginning of time period \(t\), the TPA computes the update message \(\delta_t = H_1(t)^{SK_{TPA}}\) according to his secret key \(SK_{TPA}\).
   b) The TPA sends the update message \(\delta_t\) to the client.
   c) The client can verify whether the update message is valid or not by the following equation.

\[
\hat{\cdot}(g, \delta_t) = \hat{\cdot}(PK_{TPA}, H_1(t)).
\]


3) CKeyUpdate algorithm
When the client gets the update message $\delta_t$ from the TPA at the beginning of time period $t$, he computes $SK_t = H_1(t) \cdot SK_{t-1} \cdot \delta_t$ as the signing secret key in time period $t$.

4) AuthGen algorithm
When the client wants to upload a file $F = \{m_i\} (1 \leq i \leq n)$ to the cloud in time period $t$, he does as follows:
a) The client selects $r \in_R Z_q^n$, and then computes $R = e^{-\gamma}$. 
b) Then the client computes the authenticators $\sigma_i = H_2(t||name,R) \cdot u_i^{m_i} \cdot SK_t$, where $name$ is the name of the file $F$.
c) The client uploads the file tag $tag = name || SSig_{sk}(name || t)$ and the set of authenticators $\Phi = \{t,R,\sigma_1,\ldots,\sigma_n\}$ along with file $F$ to the cloud.

5) ProofGen algorithm
a) The TPA first verifies the validity of the file tag by checking whether $SSig_{sk}(name || t)$ is a valid signature using $spk$. If it is, then the TPA selects a $c$-element subset $I = \{s_1,s_2,\ldots,s_c\}$ of set $[1,n]$ as the index of the blocks to be checked.
b) For each $i \in I$, the TPA selects random values $v_i(|| || |g|)$, and sends the challenge $\{v_i\}_{i \in I}$ to the cloud.
c) After the cloud receives the challenge $\{v_i\}_{i \in I}$, he computes an aggregate authenticator $\sigma = \prod_{i \in I} \sigma_{i,v_i}$. He also computes $\mu = \sum_{i \in I} m_i v_i$.
d) The cloud sends $P = (t,R,\sigma,\mu)$ to the TPA as his reply.

6) ProofVerify algorithm
When the TPA receives the proof $P$, he verifies whether the following equation holds:
\[
\hat{e}(g,\sigma) = \hat{e}(R, \prod_{i \in I} H_2(t||name,R)v_i \cdot u_i^\mu) \\
\cdot \hat{e}(PK_c \cdot PK_{TPA}, H_1(t) \cdot \sum_{i \in I} \mu).
\]
If it holds, then return “true”; Otherwise, return “false”.

IV. Security and Performance
A. Security Analysis

Theorem 1 (Correctness): For one random challenge $\{v_i\}_{i \in I}$ and one valid proof $P = (t,R,\sigma,\mu)$, the ProofVerify algorithm always returns “true”.

Proof. It is because the following equations hold:
\[
\hat{e}(g,\sigma) = \hat{e}(g, \prod_{i \in I} \sigma_{i,v_i}) \\
= \hat{e}(g, \prod_{i \in I} (H_2(t||name,R) \cdot u_i^{m_i} \cdot SK_t)v_i) \\
= \hat{e}(g, \prod_{i \in I} (H_2(t||name,R) \cdot u_i^{m_i} \cdot H_1(t) \cdot SK_t \cdot \delta_i)v_i) \\
= \hat{e}(g, \prod_{i \in I} H_2(t||name,R)v_i \cdot u_i \sum_{i \in I} m_i v_i^r) \\
\cdot H_1(t) \cdot SK_t \cdot SK_{TPA} \cdot \sum_{i \in I} \mu).
\]

Theorem 2 (Strong key-exposure resilience): If the CDH problem in $G_1$ is hard, then our proposed auditing scheme is strong key-exposure resilient.

Proof. We define a series of games, and analyze the difference in adversary behavior between successive games.

Game 0. Game 0 is the game defined in Section 2.

Game 1. Game 1 is the same as Game 0, with only one difference. The challenger keeps a list of all signed tags he generates. The challenger will abort when the adversary submits one valid tag generated by $SSig$ signature algorithm but not signed by the challenger.

Analysis. If the adversary makes the challenger abort in Game 1, it is easy to use the adversary to construct an attacker to break the $SSig$ signature scheme. From now on, we can assure that $name$ and $t$ in the interactions with the adversary are all generated by the challenger.

Game 2. Game 2 is the same as Game 1, with only one difference. The challenger keeps a list of his responses to the adversary’s authenticator queries. If the adversary succeeds but $R$ in its proof $P$ is different from the real $R$ in the list of authenticators that the challenger has kept, then the challenger will abort.

Analysis. We will show if the adversary can make the challenger abort in Game 2, we can construct a simulator to solve the CDH problem with non-negligible probability. The simulator acts like the challenger in Game 1, with the following differences:

Firstly, the challenger $C$ is given a CDH challenge $(g,X = g^a, Y = g^b)$. The challenger $C$ will compute $g^{ab}$ by running the adversary $A$. The challenger $C$ selects $(spk, ssk)$ as the public key and the secret key to generate a signature for file name and time period. Assume that the adversary $A$ makes $q_k$ secret key queries and $q_s$ authenticator queries.

(1) Setup phase
The challenger $C$ randomly selects $SK_c \in Z_q^*$ and sets $PK_{TPA} = X$, $PK_c = g^{SK_c}$. The challenger $C$ randomly selects $\alpha \in Z_q^*$, and sets $u = g^a$. Finally, $C$ sends the public key $PK = (g,u,PK_{TPA},PK_c,spk)$ to $A$.

(2) Query phase
$H_1$ and $H_2$ are viewed as two random oracles controlled and stored by the challenger $C$. The challenger needs to answer the queries for these random oracles from $A$.

(i) $H_1$ oracle query.

The challenger $C$ maintains an $H_1$-table to reply $A$’s queries of $H_1$ oracle. Firstly, $C$ initially sets this table empty. When $A$ queries $H_1$ oracle at $< t >$, $C$ does as follows.

(1) If $H_1$-table contains a tuple $(t,c,h_1)$ for input $< t >$, then $C$ responds with $h_1$.

(2) Otherwise, $C$ flips a coin $c \in \{0,1\}$ that yields 0 with probability $\frac{q_s}{q_k + q_s + 1}$ and 1 with probability $\frac{q_k + q_s}{q_k + q_s + 1}$. When $c = 0$, $C$ selects $\lambda \in_R Z_q^*$ and computes $g^{\lambda}$. He adds $(t,0,\lambda,h_1 = g^{\lambda})$ to $H_1$-table. Otherwise, $C$ selects $\lambda \in_R Z_q^*$.
and computes $Y^\lambda$. He adds $(t \cdot 1, \lambda, h_1 = Y^\lambda)$ to $H_1$-table. Finally, $C$ responds with $h_1$.

(II) $H_2$ oracle query. The challenger $C$ maintains an $H_2$-table to reply $A$'s queries of $H_2$ oracle. Firstly, $C$ initially sets this table empty. When $A$ queries the oracle $H_2$ at $< t || i || \text{name, R, } \varphi, h_2 >$, $C$ does as follows.

(1) If $H_2$-table contains a tuple $(< t || i || \text{name, R, } \varphi, h_2 >, R, \varphi, h_2)$ for input $< t || i || \text{name, R, } \varphi >$, then $C$ responds with $h_2$.

(2) Otherwise, $C$ selects $\varphi \in R$ and adds $(< t || i || \text{name, R, } \varphi, h_2 > = g^\varphi)$ to $H_2$-table. $C$ responds with $h_2$.

(III) The secret key query.

When $A$ queries the secret key in time period $t$, $C$ retrieves $(t, c, \lambda, h_1 = g^\lambda)$ from $H_1$-table (WLOG, assume that $A$ has the oracle $H_1$ at $t < t_>)$.

(1) If $c = 1$, $C$ aborts (Denote this event by $E_1$).

(2) Otherwise, $C$ computes $SK_t = X^{\lambda} \cdot h_1^{SK_c}$. Note that $SK_t = h_1^{SK_{TPA}} \cdot h_1^{SK_c} = g^{\lambda SK_{TPA}} \cdot h_1^{SK_c} = X^{\lambda} \cdot h_1^{SK_c}$. Finally, $C$ responds with $(SK_c, SK_t)$.

(IV) Authenticator query. When $A$ queries an authenticator at $< t, m_1, ..., m_n >$, $C$ does as follows.

(1) Firstly, $C$ recovers $(t, c, \lambda, h_1)$ from $H_1$-table.

(2) If $c = 1$, $C$ aborts (Denote this by $E_2$).

(3) Otherwise, $C$ selects $\tau \in RZ_\theta^n$, and then computes $R = g^\tau$. $C$ also selects $\sigma_{g,G_1}$.

(4) $C$ defines the hash value $H_2(< t || i || \text{name, R})$ as

$$\sigma_t \cdot X^{\lambda} \cdot h_1^{SK_c} \cdot \tau^{-1} \cdot u^{\mu_1}.$$

Note that

$$H_2(< t || i || \text{name, R}) = \left((\sigma_t \cdot X^{\lambda} \cdot h_1^{SK_c} \cdot \tau^{-1} \cdot u^{\mu_1}) \cdot r^{\mu_2} \cdot h_1^{SK_{TPA}} \cdot h_1^{SK_c} = \sigma_t \cdot X^{\lambda} \cdot h_1^{SK_c} \cdot g^{\lambda SK_{TPA}} \cdot h_1^{SK_c} = \sigma_t \cdot X^{\lambda} \cdot (g^{\lambda SK_{TPA}})^{\tau^{-1}} = \sigma_t \right).$$

Finally, $C$ responds with $(t, R, \sigma_t)$.

(3) Challenge Phase

The challenger $C$ selects a time period $t^*$ in which $A$ does not make the secret key query. $C$ sends $A$ a challenge \textbf{Chal} $= \{i, v_i\} \in \mathcal{I}$ (where $\mathcal{I} = \{s_1, s_2, ..., s_c\}$, $1 \leq s_1 \leq n$, $1 \leq l \leq c$, $1 \leq e \leq n$) and the time period $t^*$. It also requests $A$ to provide a proof $P$ of file $F = (m_1, ..., m_n)$ under \textbf{Chal} for the blocks $m_{s_1}, ..., m_{s_e}$ in time period $t^*$.

(4) Forgery phase

Finally, the adversary $A$ outputs a proof $P = (t^*, R, \sigma, \mu)$. $C$ recovers $(t^*, c^*, X^{\lambda}, h_1^*)$ from $H_1$-table.

(1) If $c^* = 0$, $C$ aborts (Denote this by $E_3$).

(2) Otherwise, if $R$ is different from the real $R$ in the authenticators set, $C$ recovers $(t^*, < i || \text{name, R, } \varphi_i, h_2 > = g^{\varphi_i})$ from $H_2$-table. In this case, $h_1^* = Y^{\lambda}$. If this forgery is valid, then

$$e(g, \sigma) = \hat{e}(R, \prod_{i \in \mathcal{I}} H_2(< t || i || \text{name, R})^{v_i} \cdot u^{\mu_i}) \cdot \hat{e}(PK_c, PK_{TPA}, H_1(t)^{\Sigma_{i \in \mathcal{I}} v_i}) = \hat{e}(R, g^{\sum_{i \in \mathcal{I}} v_i} \cdot g^{\mu_i}) \cdot \hat{e}(g^{SK_c}, X, Y^{\lambda} \cdot \sum_{i \in \mathcal{I}} v_i) = \hat{e}(g, P_{SK_c} \cdot \prod_{i \in \mathcal{I}} v_i) \cdot \hat{e}(g, Y^{\lambda} \cdot SK_c \cdot \sum_{i \in \mathcal{I}} v_i) \cdot \hat{e}(X, Y^{\lambda} \cdot \sum_{i \in \mathcal{I}} v_i).$$

So

$$e'(X, Y^{\lambda} \cdot \sum_{i \in \mathcal{I}} v_i) = \hat{e}(g, \sigma) \cdot \hat{e}(g, R^{-\alpha u + \sum_{i \in \mathcal{I}} v_i} \cdot \sum_{i \in \mathcal{I}} v_i).$$

(1) If $\sum_{i \in \mathcal{I}} v_i = 0$, $C$ aborts (Denote it by $E_4$).

(2) Otherwise, $C$ can compute $g^{\delta b}$ as

$$g^{\delta b} = (r \cdot R^{-\alpha u + \sum_{i \in \mathcal{I}} v_i}) \cdot Y^{\lambda SK_c} \cdot \sum_{i \in \mathcal{I}} v_i \cdot \sum_{i \in \mathcal{I}} v_i^{-1}.$$
As we have analyzed, there is only negligible difference probability between these games. Note that the hardness of the CDH problem implies the hardness of the discrete logarithm problem. If the CDH problem in $G_1$ is hard and the signature scheme $S$ is used for file tags is existentially unforgeable, the challenger will reject except when the prover responds with correctly computed values in proof $P$.

Now we will construct a knowledge extractor to extract the whole challenged file blocks $m_{t_1}, m_{t_2}, ..., m_{t_n}$. By selecting independent coefficients $v_1, v_2, ..., v_n$ to execute the challenge phase on the blocks $m_{t_1}, m_{t_2}, ..., m_{t_n}$ for $c$ times, the extractor can collect $c$ independent linear equations in the variables $m_{t_1}, m_{t_2}, ..., m_{t_n}$. The extractor can extract $m_{t_1}, m_{t_2}, ..., m_{t_n}$ by solving these equations. This completes the proof of Theorem 2.

Theorem 3 (Detectability): Our proposed auditing scheme is $(\frac{m}{n} - 1 - \frac{(n-m)c}{n})$ detectable if the cloud stores a file with $n$ blocks including $m$ bad (deleted or modified) blocks, and $c$ blocks are challenged.

Assume that the cloud stores a file with total $n$ blocks including $m$ bad (deleted or modified) blocks. The number of challenged blocks is $c$. Thus, the bad blocks can be found out if and only if at least one of the challenged blocks chosen by the TPA matches the bad blocks. We use a discrete random variable $X$ to denote the number of blocks selected by the challenger that matches the block-tag pairs changed by the adversary. We use $P_X$ to denote the probability that at least one block chosen by the challenger matches the blocks changed by the adversary. So

$$P_X = P(X \geq 1) = 1 - P(X = 0) = 1 - \frac{n-m}{n-1} \times \cdots \times \frac{n-c+1-m}{n-c+1}$$

We can get $P_X \geq 1 - (\frac{n-m}{n})^c$. Thus, the proposed auditing scheme is $(\frac{m}{n} - 1 - \frac{(n-m)c}{n})$ detectable if the cloud stores a file with $n$ blocks including $m$ bad (deleted or modified) blocks, and $c$ blocks are challenged.

Theorem 4: The TPA cannot forge the authenticator for any block through his secret key and the update messages he generates.

Proof. From AuthGen algorithm, we know the computation of any authenticator in time period $t$ requires the secret key $SK_t$. According to the key update of the client, the secret key $SK_t$ is determined by two secret keys $SK_{TP_A}$ and $SK_c$, that is, $SK_t = H_1(t)^{SK_c} \cdot \delta_t = H_1(t)^{SK_c} \cdot H_1(t)^{SK_{TP_A}}$. From the view of the TPA in our scheme, the TPA only knows $SK_{TP_A}$ and can only compute the update message $\delta_t$. It means the TPA cannot compute $SK_t$ because it does not know the secret key $SK_c$ of the client. Therefore, the TPA cannot forge the authenticator for any block through his secret key and the update messages he generates.

Theorem 5: The proposed strong key-exposure resilient auditing for secure cloud storage is verifiable.

Proof. From UMGen algorithm, we know any incorrect update key cannot pass the equation $\hat{e}(g, \delta_t) = \hat{e}(PK_{TP_A}, H_1(t))$. Therefore, this Theorem follows.
B. Performance Analysis

We firstly give the numerical analysis of computation and communication overhead of the proposed scheme in main phases, and then evaluate the proposed scheme through several practical experiments.

(1) Computation and Communication Overhead

In the phase of key update, the client needs to compute the signing secret key $SK_t$, which costs $Exp_{G_1} + Mul_{G_1}$, where $Exp_{G_1}$ denotes the computation of one exponentiation in $G_1$ and $Mul_{G_1}$ denotes the computation of one multiplication in $G_1$. In the phase of challenge generation, the TPA only chooses some random values to construct a challenge message, which costs little computation. In the phase of proof generation, the cloud needs to generate the proof $P = (t, R, \sigma, \mu)$. It costs $c \cdot Exp_{G_1} + (c-1) \cdot Mul_{G_1} + c \cdot Mul_{Z_q} + (c-1) \cdot Sum_{Z_q}$, where $Mul_{Z_q}$ denotes the computation of one multiplication and $Sum_{Z_q}$ denotes the computation of one addition in $Z_q$. In the phase of proof verifying, the cloud needs to verify whether the proof $P$ is valid or not by the verification equation. It costs $(c+2) \cdot Exp_{G_1} + (c+1) \cdot Mul_{G_1} + (c-1) \cdot Sum_{Z_q} + 3Pair + Mul_{G_2}$, where $Pair$ denotes the computation of one pairing, and $Mul_{G_2}$ denotes the computation of one multiplication in $G_2$.

The communication overhead in our proposed scheme includes the challenge overhead and the proof overhead. The challenge message is composed by $c$ pairs of elements $(i, v_i)$. So the challenge overhead is $c \cdot (|n| + |q|)$, where $|n|$ is the length of an index and $|q|$ is the length of an element in $Z_q$.

The overhead of proof message $(t, R, \sigma, \mu)$ is about $2|G_1| + |q|$, where $|G_1|$ is the length of an element in $G_1$ (the size of $t$ is negligible compared with $G_1$ and $q$).

(2) Experimental Results

With the help of the Pairing-Based Cryptography (PBC) library [30], we evaluate the proposed scheme in several experiments. We run these experiments on a Linux server with Intel processor running at 2.70 GHz and 4 GB memory. We choose a bilinear map that uses a supersingular elliptic curve to achieve the fast pairing operations. The base field of this curve is 160 bits, the size of an element in $\mathbb{Z}_q^*$ is 20 bytes, and the size of an element in group $G_1$ is 128 bytes. In our experiments, the data file is set to 20M, which consists of 1,000,000 blocks. In order to compare the efficiency of our proposed scheme with that of the scheme [13] which uses a full binary tree with depth 2, we consider the current time period varying from 0 to 14.
In our proposed scheme, the time for the client to update a secret key is independent of the time period because it needs one exponentiation and one multiplication in $G_1$ in any time period. In contrast, the key update time of the client is relevant to the time period in scheme [13]. We compare the key update time between the both schemes in Fig.3. In the scheme [13], the key update time is related to the depth of the node corresponding to the current time period in the full binary tree. When the node is the internal node, the key update time is about 12.6ms; when the node is the leaf node, the key update time is nearly 0ms. In our proposed scheme, the key update time is about 5.8ms in all time periods. It means the client can update the signing secret key using the same computational resource in each time period. The average update cost of our proposed scheme is the same as that in the scheme [13]. Therefore, our proposed scheme achieves stronger security without decreasing the efficiency of key updates for the client. In addition, our proposed scheme supports key updates for unlimited time periods. However, the lifetime of the file stored in cloud must be known and fixed initially in the scheme [13]. Thus, the scheme [13] can only support key updates for a fixed $T$ time periods. When $T$ time periods are over, the cloud storage auditing cannot work any longer.

With the different number of challenged blocks, we demonstrate the time of different auditing processes including the challenge generation process, the proof generation process, and the proof verification process through the experiments in Fig.4. In our experiment, the number of challenged blocks varies from 100 to 1,000. From Fig.4, we can know the challenge generation process spends the least time, which is 0.26s at most; the proof generation process spends a little more time, varying from 0.18s to 1.79s; the proof generation process spends the most time, varying from 0.22s to 2.05s.

In our proposed scheme, the communication overhead consists of the challenge overhead and the proof overhead. As same as most of cloud storage auditing schemes, the challenge overheads of the proposed scheme and the scheme [13] are both $Chal = \sum_{i \in \mathbb{N}} v_i$, where $i$ is used to determine which blocks will be checked and $v_i$ is used to mix the challenged blocks. Therefore, the proposed scheme and the scheme [13] have the same challenge overhead. Because the challenge overhead is $c \cdot (|\mathbb{N}| + |\mathbb{Q}|)$, it has a linear relationship with the number $c$ of challenged blocks. In Fig.5, when the number of checked blocks varies from 100 to 1,000, the size of the challenge message varies from 2.29KB to 22.89KB. Because the size of the proof message is $2|G_1| + |\mathbb{Q}|$, it is independent of the time period. From Fig.6, we can see that the size of the proof message keeps about 148B in all the time periods, which is much smaller than that in the scheme [13].

V. Conclusion

In this paper, we further study on how to deal with the key exposure problem in cloud storage auditing. We propose a new paradigm called strong key-exposure resilient auditing scheme for secure cloud storage. In this paradigm, the security of the cloud storage auditing not only earlier than but also later than the key exposure can be preserved. We formalize the definition and the security model of this new kind of cloud storage auditing and design a concrete scheme. The security proof and the experimental results demonstrate that the proposed scheme is secure and efficient.

REFERENCES


Jia Yu is a professor of the College of Computer Science and Technology and the department director of Information Security at Qingdao University. He received the M.S. and B.S. degrees in School of Computer Science and Technology from Shandong University in 2003 and 2000, respectively. He received Ph. D. degree in Institute of Network Security from Shandong University, in 2006. He was a visiting professor with the Department of Computer Science and Engineering, the State University of New York at Buffalo, from Nov. 2013 to Nov. 2014. His research interests include cloud computing security, key evolving cryptography, digital signature, and network security.

Huaqun Wang received the BS degree in mathematics education from the Shandong Normal University and the MS degree in applied mathematics from the East China Normal University, both in China, in 1997 and 2000, respectively. He received the Ph.D. degree in information security from Nanjing University of Posts and Telecommunications in 2006. He is currently a professor of Nanjing University of Posts and Telecommunications, China. His research interests include applied cryptography, network security, and cloud computing security.