Enabling Cloud Storage Auditing with Key-Exposure Resistance

Jia Yu, Kui Ren, *Senior Member, IEEE,* Cong Wang, *Member, IEEE* and Vijay Varadharajan, *Senior Member, IEEE*

***Abstract*—Cloud storage auditing is viewed as an important service to verify the integrity of the data in public cloud. Current auditing protocols are all based on the assumption that the client's secret key for auditing is absolutely secure. However, such assumption may not always be held, due to the possibly weak sense of security and/or low security settings at the client. If such a secret key for auditing is exposed, most of the current auditing protocols would inevitably become unable to work. In this paper, we focus on this new aspect of cloud storage auditing. We investigate how to reduce the damage of the client's key exposure in cloud storage auditing, and give the first practical solution for this new problem setting. We formalize the definition and the security model of auditing protocol with key-exposure resilience and propose such a protocol. In our design, we employ the binary tree structure and the pre-order traversal technique to update the secret keys for the client. We also develop a novel authenticator construction to support the forward security and the property of blockless verifiability. The security proof and the performance analysis show that our proposed protocol is secure and efficient.**

***Index Terms*—Data storage, cloud storage auditing, homomorphic linear authenticator, cloud computation, key-exposure resistance**

I. Introduction

CLOUD storage auditing is used to verify the integrity of the data stored in public cloud, which is one of the important security techniques in cloud storage. In recent years, auditing protocols for cloud storage have attracted much attention and have been researched intensively [1–16]. These protocols focus on several different aspects of auditing, and how to achieve high bandwidth and computation efficiency is one of the essential concerns. For that purpose, the Homomorphic Linear Authenticator (HLA) technique that supports blockless verification is explored to reduce the overheads of computation and communication in auditing protocols, which allows the auditor to verify the integrity of the data in cloud without retrieving the whole data. Many cloud storage auditing protocols like [1–8] have been proposed based on this technique. The privacy protection of data is also an important aspect of cloud storage auditing. In order to reduce the computational burden of the client, a third-party auditor (TPA) is introduced to help the client to periodically check the integrity of the data in cloud. However, it is possible for the TPA to get the client's data after it executes the auditing protocol multiple times. Auditing protocols in [9, 10] are designed to ensure the privacy of the client's data in cloud. Another aspect having been addressed in cloud storage auditing is how to support data dynamic operations. Wang *et al.* [11] have proposed an auditing protocol supporting fully dynamic data operations including modification, insertion and deletion. Auditing protocols in [2, 9, 12, 13] can also support dynamic data operations. Other aspects, such as proxy auditing [14], user revocation [15] and eliminating certificate management [16] in cloud storage auditing have also been studied. Though many research works about cloud storage auditing have been done in recent years, a critical security problem—the key exposure problem for cloud storage auditing, has remained unexplored in previous researches. While all existing protocols focus on the faults or dishonesty of the cloud, they have overlooked the possible weak sense of security and/or low security settings at the client.

In fact, the client's secret key for cloud storage auditing may be exposed, even known by the cloud, due to several reasons. Firstly, the key management is a very complex procedure [17] which involves many factors including system policy, user training, etc. One client often needs to manage varieties of keys to complete different security tasks. Any careless mistake or fault in managing these keys would make the key exposure possible. It is not uncommon to see a client choosing to use cheap software-based key management for economical factors [18], which may only provide limited protection and make the sensitive secret keys vulnerable to exposure. Secondly, the client himself may be the target and vulnerable to many Internet based security attacks. For an ordinary client, the sense of security protection can be relatively weaker, compared with the case of enterprises and organizations. Hence, it is possible for a client to unintentionally download malicious software from Internet [19, 20] or to overlook the timely security patch to their computer system. Both of these cases could give the hacker easy access

This research is supported by National Natural Science Foundation of China (61272425,61402245), US National Science Foundation (CNS-1226677), Research Grants Council of Hong Kong (CityU 1385130), and Shandong provincial Key Laboratory of Computer Network (SDKLCN-2013-03). (*Corresponding author: J. Yu*)

J. Yu is with the College of Information Engineering, Qingdao University, Qingdao 266071, China and with Shandong provincial Key Laboratory of Computer Network, Jinan 250014, China. He is also a visiting professor with the Department of Computer Science and Engineering, The State University of New York at Buffalo. E-mail:yujia@qdu.edu.cn.

K. Ren is with the Department of Computer Science and Engineering, The State University of New York at Buffalo. E-mail:kuiren@buffalo.edu.

C. Wang is with the Department of Computer Science, City University of Hong Kong. E-mail:congwang@cityu.edu.hk.

V. Varadharajan is with the Department of Computing, Macquarie University, Australia. E-mail:vijay.varadharajan@mq.edu.au.

to their secret keys. Last but not the least, the cloud also has incentives to get clients' secret keys for storage auditing, e.g., through trading with the aforementioned hackers. Specifically, if the cloud gets these keys, it can regenerate the fake data and forge their authenticators to easily hide the data loss incidents, e.g., caused by Byzantine failures, from the client, while maintaining its reputation [9, 10]. In the malicious case, it can even discard the client's data that are rarely accessed to save the storage space [10–12], without worrying about failure to pass the auditing protocol initiated by the client. Obviously, the auditing secret key exposure could be disastrous for the clients of cloud storage applications.

Therefore, how to deal with the client's secret key exposure for cloud storage auditing is a very important problem. Unfortunately, previous auditing protocols did not consider this critical issue, and any exposure of the client's secret auditing key would make most of the existing auditing protocols unable to work correctly. In this paper, we focus on how to reduce the damage of the clients key exposure in cloud storage auditing. Our goal is to design a cloud storage auditing protocol with built-in key-exposure resilience. How to do it efficiently under this new problem setting brings in many new challenges to be addressed below. First of all, applying the traditional solution of key revocation to cloud storage auditing is not practical. This is because, whenever the client's secret key for auditing is exposed, the client needs to produce a new pair of public key and secret key and regenerate the authenticators for the client's data previously stored in cloud. The process involves the downloading of whole data from the cloud, producing new authenticators, and re-uploading everything back to the cloud, all of which can be tedious and cumbersome. Besides, it cannot always guarantee that the cloud provides real data when the client regenerates new authenticators. Secondly, directly adopting standard key-evolving technique [21–23] is also not suitable for the new problem setting. It can lead to retrieving all of the actual files blocks when the verification is proceeded. This is partly because the technique is incompatible with blockless verification. The resulting authenticators cannot be aggregated, leading to unacceptably high computation and communication cost for the storage auditing.

Contribution. The contribution of this paper can be summarized as follows:

(1) We initiate the first study on how to achieve the key-exposure resilience in the storage auditing protocol and propose a new concept called auditing protocol with key-exposure resilience. In such a protocol, any dishonest behaviors, such as deleting or modifying some client's data stored in cloud in previous time periods, can all be detected, even if the cloud gets the client's current secret key for cloud storage auditing. This very important issue is not addressed before by previous auditing protocol designs. We further formalize the definition and the security model of auditing protocol with key-exposure resilience for secure cloud storage.

(2) We design and realize the first practical auditing protocol with built-in key-exposure resilience for cloud storage. In order to achieve our goal, we employ the binary tree structure, seen in a few previous works [24–28] on different cryptographic designs, to update the secret keys of the client. Such a binary tree structure can be considered as a variant of the tree structure used in the HIBE scheme [29]. In addition, the pre-order traversal technique is used to associate each node of a binary tree with each time period. In our detailed protocol, the stack structure is used to realize the pre-order traversal of the binary tree. We also design a novel authenticator supporting the forward security and the property of blockless verifiability.

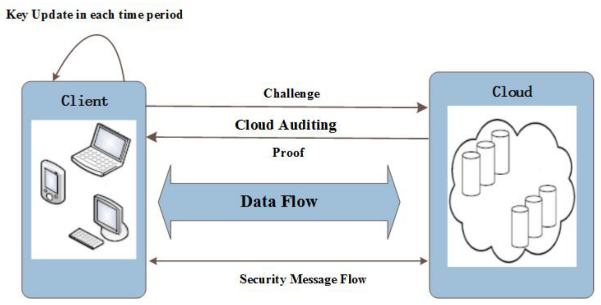


Fig. 1. System model of our cloud storage auditing

(3) We prove the security of our protocol in the formalized security model, and justify its performance via concrete asymptotic analysis. Indeed, the proposed protocol only adds reasonable overhead to achieve the key-exposure resilience. We also show that our proposed design can be extended to support the TPA, lazy update and multiple sectors.

Organization. The rest paper is organized as follows: In Section 2, we introduce the system model, definition, security model and preliminaries of our work. Then, we give a concrete description of our protocol in Section 3. The efficiency evaluation and security theorem are given in Section 4. Section 5 presents further discussions. We conclude the paper in Section 6. The detailed security proof is shown in the appendix.

II. Formulation and preliminaries

A. System model

We show an auditing system for secure cloud storage in Fig. 1. The system involves two parties: the client (files owner) and the cloud. The client produces files and uploads these files along with corresponding authenticators to the cloud. The cloud stores these files for the client and provides download service if the client requires. Each file is furthermore divided into multiple blocks [2]. For the simplicity of description, we assume that the client also plays the role of auditor in our system (In fact, our protocol completely supports for the third party auditor, which will be discussed in Section 5). The client can periodically audit whether his files in cloud are correct. The lifetime of files stored in the cloud is divided into $T + 1$ time periods (from 0-th to T-th time periods). In our model, the client will update his secret keys for cloud storage auditing in the end of each time period, but the public key is always unchanged. The cloud is allowed to get the client's secret key for cloud storage auditing in one certain time period. It means the secret key exposure can happen in this system model.

B. Definition and Security Model

(1) The definition of auditing protocol with key-exposure resilience

Definition 1 (Auditing protocol with key-exposure resilience): An auditing protocol with key-exposure resilience is composed by five algorithms (*SysSetup*, *KeyUpdate*, *AuthGen*, *ProofGen*, *ProofVerify*), shown below:

1) $SysSetup(1^k, T) \rightarrow (PK, SK_0)$: the system setup algorithm is a probabilistic algorithm which takes as input a security parameter k and the total number of time periods T, and generates a public key PK and the initial client's secret key SK_0 . This algorithm is run by the client.
2) $KeyUpdate(PK, j, SK_j) \rightarrow (SK_{j+1})$: the key update algorithm is a probabilistic algorithm which takes as input the public key PK, the current period j and a client's secret key SK_j, and generates a new secret key SK_{j+1} for the next period $j + 1$. This algorithm is run by the client.
3) $AuthGen(PK, j, SK_j, F) \rightarrow (\Phi)$: the authenticator generation algorithm is a probabilistic algorithm which takes as input the public key PK, the current period j, a client's secret key SK_j and a file F, and generates the set of authenticators Φ for F in time period j. This algorithm is also run by the client.
4) $ProofGen(PK, j, Chal, F, \Phi) \rightarrow (P)$: the proof generation algorithm is a probabilistic algorithm which takes as input the public key PK, a time period j, a challenge $Chal$, a file F and the set of authenticators Φ, and generates a proof P which means the cloud possesses F. Here, $(j, Chal)$ pair is issued by the auditor, and then used by the cloud. This algorithm is run by the cloud.
5) $ProofVerify(PK, j, Chal, P) \rightarrow$ ("$True$" or "$False$"): the proof verifying algorithm is a deterministic algorithm which takes as input the public key PK, a time period j, a challenge $Chal$ and a proof P, and returns "$True$" or "$False$". This algorithm is run by the client.

(2)Security model

Our security model considers the notion of the forward security [21] and data possession property [1]. In Table 1, we indicate a game to describe an adversary $\mathcal{A}$ against the security of an auditing protocol with key-exposure resilience. Specifically, above game is composed of the following phases:

TABLE I
A GAME TO DESCRIBE AN ADVERSARY $\mathcal{A}$ AGAINST THE SECURITY OF THE PROTOCOL

```
Game Adversary-Forge($A$)
$j = 0; A \overset{PK}{\leftarrow} SysSetup(\cdot);$
Repeat
    $A \leftarrow AuthGen_{SK_j}(F = (m_1, ..., m_n));$
    $SK_{j+1} \leftarrow KeyUpdate(SK_j); j \leftarrow j + 1;$
Until $A$ goes to the break-in phase;
$b = j; A \leftarrow SK_b;$
$A \leftarrow (Chal, j^* (j^* < b));$
$P = A(Forge, F, Chal, j^* (j^* < b));$
if ProofVerify($PK, j^*, Chal, P$) = 1
then return "True" else return "False".
```

1) **Setup Phase**. The challenger runs the $SysSetup$ algorithm to generate the public key PK and the initial client's secret key SK_0. The challenger sends PK to an adversary $\mathcal{A}$ and holds SK_0 himself. Set time period $j = 0$.
2) **Query Phase**. $\mathcal{A}$ running in this phase can adaptively query as follows.
Authenticator Queries. $\mathcal{A}$ can query the authenticators of the blocks it selects in time period j. It can adaptively select a series of blocks $m_1,\cdots,m_n$, and sends them to the challenger. The challenger computes the authenticators for $m_i (i = 1, \cdots, n)$ in time period j, and sends them back to $\mathcal{A}$. $\mathcal{A}$ stores all blocks $F = (m_1, \cdots, m_n)$ and their corresponding authenticators.
Set time period $j = j + 1$.
At the end of each time period, $\mathcal{A}$ can select to still stay in query phase or go to the break-in phase.
3) **Break-in Phase**. This phase models the possibility of key exposure. Set the break-in time period $b = j$. The challenger generates the secret key SK_b by the $KeyUpdate$ algorithm and sends it to $\mathcal{A}$. **Challenge Phase**. The challenger sends $\mathcal{A}$ a challenge $Chal$ and a time period $j^* (j^* < b)$. He also requests the adversary to provide a proof of possession for the blocks $m_{s_1},\cdots,m_{s_c}$ of file $F = (m_1,\cdots,m_n)$ under $Chal$ in time period j^* , where $1 \leq s_l \leq n$, $1 \leq l \leq c$, and $1 \leq c \leq n$.
4) **Forgery Phase**. $\mathcal{A}$ outputs a proof of possession P for the blocks indicated by $Chal$ in time period j^*, and returns P. If $ProofVerify(PK, j^*, Chal, P)$="$True$", then $\mathcal{A}$ wins in above game.

The above security model captures that an adversary cannot forge a valid proof for a time period prior to key exposure without owning all the blocks corresponding to a given challenge, if it cannot guess all the missing blocks. In each time period prior to key exposure, the adversary is allowed to query the authenticators of all the blocks. The adversary can be given a secret key for auditing in the key-exposure (break-in) time period. Obviously, the adversary does not need to query the authenticators in or after the key-exposure time period because it can compute all secret keys after this time period using the exposed secret key. The goal of the adversary is to construct a valid proof of possession P for the blocks indicated by $Chal$ in time period j^*. Definition 2 shows that there exists a knowledge extractor allowing the extraction of the challenged file blocks whenever $\mathcal{A}$ can produce a valid proof of possession P in time period j^*. Definition 3 describes the detectability for auditing protocol, which guarantees the cloud maintains the blocks that are not challenged with high probability.

Definition 2: (Key exposure resistance) We say an auditing protocol is key exposure resistant if the following condition holds: whenever an adversary $\mathcal{A}$ in above game that can cause the challenger to accept its proof with non-negligible probability, there exists an efficient knowledge extractor that can extract the challenged file blocks except possibly with negligible probability.

Definition 3 (Detectability): We say an auditing protocol is (ρ, δ) detectable $(0 < \rho, \delta < 1)$ if, given a fraction ρ of corrupted blocks, the probability that the corrupted blocks are detected is at least δ.

C. Preliminaries

Definition 4 (Bilinear Pairing): There are two multiplicative cyclic groups G_1 and G_2, which have the same prime order q. We say a map $\hat{e} : G_1 \times G_1 \rightarrow G_2$ is a bilinear pairing if it satisfies:

1) Bilinearity: For $\forall g_1, g_2 \in G_1$ and $\forall a, b \in Z_q^*$, $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$.
2) Non-degeneracy: Whenever g is a generator of G_1, $\hat{e}(g, g) \neq 1$.
3) Computability: There is an efficient algorithm to compute $\hat{e}(g_1, g_2)$ for $\forall g_1, g_2 \in G_1$.

Definition 5 (CDH problem): Given g^a and g^b, where g is a generator of G_1 and $a, b \in_R Z_q^*$, compute g^{ab}.

An algorithm $\mathcal{IG}$ is called CDH parameter generator if it takes as input a security parameter k, and outputs the description of two groups G_1, G_2 and a bilinear map $\hat{e} : G_1 \times G_1 \rightarrow G_2$ satisfying that the CDH problem in G_1 is difficult.

III. Our Proposed Protocol

We firstly show two basic solutions for the key-exposure problem of cloud storage auditing before we give our core protocol. The first is a naive solution, which in fact cannot fundamentally solve this problem. The second is a slightly better solution, which can solve this problem but has a large overhead. They are both impractical when applied in realistic settings. And then we give our core protocol that is much more efficient than both of the basic solutions.

A. Naive Solution

In this solution, the client still uses the traditional key revocation method. Once the client knows his secret key for cloud storage auditing is exposed, he will revoke this secret key and the corresponding public key. Meanwhile, he generates one new pair of secret key and public key, and publishes the new public key by the certificate update. The authenticators of the data previously stored in cloud, however, all need to be updated because the old secret key is no longer secure. Thus, the client needs to download all his previously stored data from the cloud, produce new authenticators for them using the new secret key, and then upload these new authenticators to the cloud. Obviously, it is a complex procedure, and consumes a lot of time and resource. Furthermore, because the cloud has known the original secret key for cloud storage auditing, it may have already changed the data blocks and the corresponding authenticators. It would become very difficult for the client to even ensure the correctness of downloaded data and the authenticators from the cloud. Therefore, simply renewing secret key and public key cannot fundamentally solve this problem in full.

B. Slightly Better Solution

The client initially generates a series of public keys and secret keys: (PK_1,SK_1), (PK_2,SK_2),$\cdots$, (PK_T,SK_T). Let the fixed public key be $(PK_1, \cdots, PK_T)$ and the secret key in time period j be $(SK_j,\cdots,\ SK_T)$. If the client uploads files to the cloud in time period j, the client uses SK_j to compute authenticators for these files. Then the client uploads files and authenticators to the cloud. When auditing these files, the client uses PK_j to verify whether the authenticators for these files are indeed generated through SK_j. When the time period changes from j to $j+1$, the client deletes SK_j from his storage. Then the new secret key is $(SK_{j+1},\cdots,SK_T)$.

This solution is clearly better than the naive solution. Note that the keys $SK_1, \cdots, SK_{j-1}$ have been deleted during or before time period j. So from this period onwards, the cloud cannot forge any authenticator uploaded in previous time periods, even if the secret key $(SK_j,\cdots,\ SK_T)$ in time period j for auditing is exposed. It means the cloud can not change the client's data and forge any authenticator that can be verified under $PK_t(t < j)$ when it gets the client's secret key in time period j. However, the drawback of this solution is the following: the public key and the secret key has to be very long and linear with the total number of possible time periods T, which is supposed to well cover the lifetime of data to be stored in the cloud. As the continuous trend of migration to cloud, it is not hard to envision the T value to be very large, making such linear overhead practically unacceptable.

C. Our Cloud Storage Auditing with Key-exposure Resilience

Our goal is to design a practical auditing protocol with key-exposure resilience, in which the operational complexities of key size, computation overhead and communication overhead should be at most sublinear to T. In order to achieve our goal, we use a binary tree structure to appoint time periods and associate periods with tree nodes by the pre-order traversal technique [24]. The secret key in each time period is organized as a stack. In each time period, the secret key is updated by a forward-secure technique [28]. It guarantees that any authenticator generated in one time period cannot be computed from the secret keys for any other time period later than this one. Besides, it helps to ensure that the complexities of keys size, computation overhead and communication overhead are only logarithmic in total number of time periods T. As a result, the auditing protocol achieves key-exposure resilience while satisfying our efficiency requirements. As we will show later, in our protocol, the client can audit the integrity of the cloud data still in aggregated manner, i.e., without retrieving the entire data from the cloud. As same as the key-evolving mechanisms[21–23], our proposed protocol does not consider the key exposure resistance during one time period. Below, we will give the detailed description of our core protocol.

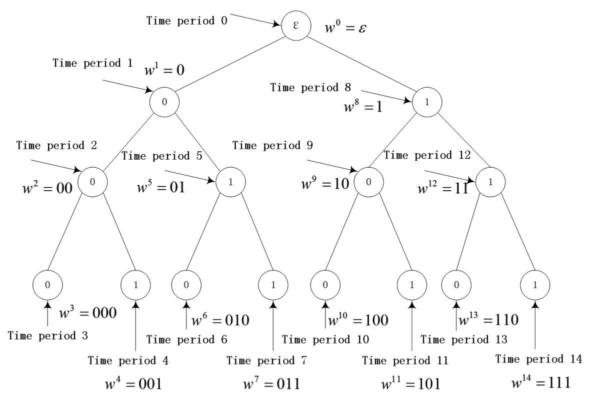


Fig. 2. An example of how to associate the nodes with time periods in a binary tree with depth 4

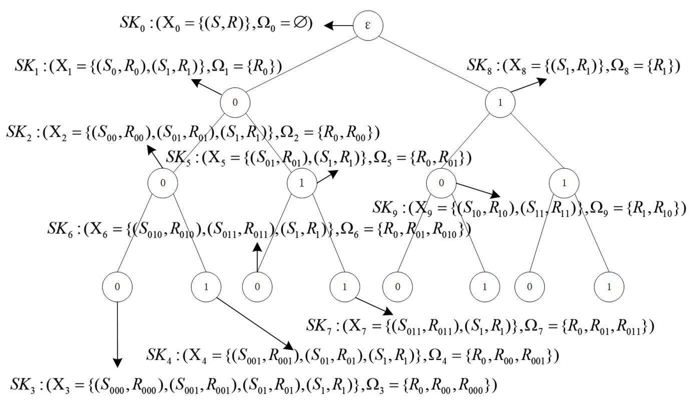

$SK_0:(X_0=\{(S,R)\},\Omega_0=\varnothing)$
$SK_1:(X_1=\{(S_0,R_0),(S_1,R_1)\},\Omega_1=\{R_0\})$
$SK_8:(X_8=\{(S_1,R_1)\},\Omega_8=\{R_1\})$
$SK_2:(X_2=\{(S_{00},R_{00}),(S_{01},R_{01}),(S_1,R_1)\},\Omega_2=\{R_0,R_{00}\})$
$SK_5:(X_5=\{(S_{01},R_{01}),(S_1,R_1)\},\Omega_5=\{R_0,R_{01}\})$
$SK_9:(X_9=\{(S_{10},R_{10}),(S_{11},R_{11})\},\Omega_9=\{R_1,R_{10}\})$
$SK_6:(X_6=\{(S_{010},R_{010}),(S_{011},R_{011}),(S_1,R_1)\},\Omega_6=\{R_0,R_{01},R_{010}\})$
$SK_7:(X_7=\{(S_{011},R_{011}),(S_1,R_1)\},\Omega_7=\{R_0,R_{01},R_{011}\})$
$SK_4:(X_4=\{(S_{001},R_{001}),(S_{01},R_{01}),(S_1,R_1)\},\Omega_4=\{R_0,R_{00},R_{001}\})$
$SK_3:(X_3=\{(S_{000},R_{000}),(S_{001},R_{001}),(S_{01},R_{01}),(S_1,R_1)\},\Omega_3=\{R_0,R_{00},R_{000}\})$

Fig. 3. An example to show what elements are included in $SK_j(0\leq\ j\leq 9)$ when $l=4$

1) Notations and Structures:

We divide the whole lifetime of data into discrete time

has been used to design many cryptographic schemes [24□ identifier name. Without loss of generality, we also assume periods 0

29], to appoint these time periods. Let each node of the tree be associated with one period, so a full binary tree with depth l can associate 2^l-1 periods (Here $T=2^l-2$). Denote the node associated with period j by a binary string $\mathbf{w^j}=w_1\cdots w_t$, where t is the bit length of $\mathbf{w^j}$. Let $\mathbf{w^j}|_k(k\leq t)$ denote a k-prefix of $\mathbf{w^j}$. For example, $\mathbf{w^6}|_2=01$ and $\mathbf{w^6}|_1=0$ if $\mathbf{w^6}=010$. Let $\mathbf{w^j}0$ and $\mathbf{w^j}1$ denote the left child node and the right child node of $\mathbf{w^j}$, and node $\mathbf{w^j}|_{\bar{k}}$ denote the sibling node of $\mathbf{w^j}|_k$. Let ε denote an empty binary string. Associate the nodes of the tree with the periods by pre-order traversal technique as follows: Begin with root node $w^0=\varepsilon$. If $\mathbf{w^j}$ is an internal node, then $\mathbf{w^{j+1}}=w'0$; if $\mathbf{w^j}$ is a leaf node, then $\mathbf{w^{j+1}}=w'1$, where w' is the longest string such that $w'0$ is a prefix of $\mathbf{w^j}$. We give an example of a binary tree with depth 4 ($l=4$) to show how to associate nodes with time periods in Fig. 2.

The public key in our protocol is denoted by PK which is fixed during the whole lifetime. In our protocol, each node of the binary tree corresponding to j has one key pair $(S_{\mathbf{w^j}},R_{\mathbf{w^j}})$, where $S_{\mathbf{w^j}}$ is called as the node secret key which is used to generate authenticators and $R_{\mathbf{w^j}}$ is called as verification value which is used to verify the validity of authenticators. The key pair of the root node is denoted by (S,R). The client's secret key SK_j in period j is composed by two parts X_j and Ω_j. The first part X_j is a set composed by the key pair $(S_{\mathbf{w^j}},R_{\mathbf{w^j}})$ and the key pairs of the right siblings of the nodes on the path from the root to $\mathbf{w^j}$. That is, if $w'0$ is a prefix of $\mathbf{w^j}$, then X_j contains the secret key $(S_{w'1},R_{w'1})$. In our protocol, the first part X_j is organized as a stack satisfying first-in first-out principle with $(S_{\mathbf{w^j}},R_{\mathbf{w^j}})$ on top. The stack is initially set (S,R) in time period 0. The second part Ω_j is composed by the verification values from the root to node $\mathbf{w^j}$ except the root. So $\Omega_j=(R_{\mathbf{w^j}|_1},\cdots,R_{\mathbf{w^j}|_t})$ when $\mathbf{w^j}=w_1\cdots w_t$. We give an example to show what elements are included in $SK_j(0\leq\ j\leq 9)$ when $l=3$ in Fig. 3.

In the following presentation, we use F to represent a file which the client wants to store in cloud. It consists of n blocks $m_i(i=1,\cdots,n)$ and is denoted by $F=(m_1,\cdots,m_n)$. In previous cloud storage auditing protocols [10, 11], a signature $SSig$ is used to ensure the integrity of the unique file

that there is a signature $SSig$ in our protocol, which is used to ensure the integrity of not only the file identifier $name$ but also the time period j and the set Ω_j for verification. We assume the client has held the secret key for signature $SSig$ and the corresponding public key has been published. Such assumption can be easily achieved in practice without much overhead, and will also simplify our protocol description thereafter.

2) Description of Our Protocol:

1) $SysSetup$: Input a security parameter k and the total time period T. Then
   a) Run $\mathcal{IG}(1^k)$ to generate two multiplicative groups G_1, G_2 of some prime order q and an admissible pairing $\hat{e}:G_1\times G_1\rightarrow G_2$.
   b) Choose cryptographic hash functions $H_1:G_1\rightarrow G_1$, $H_2:\{0,1\}^*\times G_1\rightarrow Z_q^*$ and $H_3:\{0,1\}^*\times G_1\rightarrow G_1$. Select two independent generators $g,u\in G_1$.
   c) The client selects $\rho\in Z_q^*$ at random, and computes $R=g^\rho$ and $S=H_1(R)^\rho$.
   d) The public key is $PK=(G_1,G_2,\hat{e},g,u,T,H_1,H_2,H_3,R)$. Set $X_0=\{(S,R)\}$ and $\Omega_0=\varnothing$ (where $\varnothing$ is null set). The initial secret key is $SK_0=(X_0,\Omega_0)$.
2) $KeyUpdate$: Input the public key PK, the current time period j and a secret key SK_j. Denote the node associated with period j with a binary string $\mathbf{w^j}=w_1\cdots w_t$.

   As we have mentioned in this section, X_j is organized as a stack which consists of $(S_{\mathbf{w^j}},R_{\mathbf{w^j}})$ and the key pairs of the right siblings of the nodes on the path from the root to $\mathbf{w^j}$. The top element of the stack is $(S_{\mathbf{w^j}},R_{\mathbf{w^j}})$. Firstly, pop $(S_{\mathbf{w^j}},R_{\mathbf{w^j}})$ off the stack. Then do as follows:
   a) If $\mathbf{w^j}$ is an internal node (Note $\mathbf{w^{j+1}}=\mathbf{w^j}0$ in this case), then select $\rho_{\mathbf{w^j}0},\rho_{\mathbf{w^j}1}\in Z_q^*$, and compute $R_{\mathbf{w^j}0}=g^{\rho_{\mathbf{w^j}0}}$, $R_{\mathbf{w^j}1}=g^{\rho_{\mathbf{w^j}1}}$,

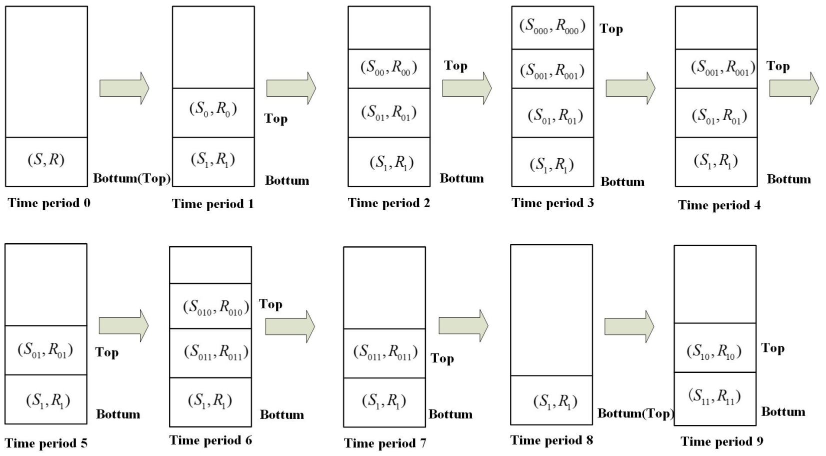


Fig. 4. An example to show the stack changes from time period 0 to time period 9 when $l = 4$

$S_{\mathbf{w^j}0} = S_{\mathbf{w^j}} \cdot H_1(R)^{\rho_{\mathbf{w^j}0} h_{\mathbf{w^j}0}}$ and $S_{\mathbf{w^j}1} = S_{\mathbf{w^j}} \cdot H_1(R)^{\rho_{\mathbf{w^j}1} h_{\mathbf{w^j}1}}$, where $h_{\mathbf{w^j}0} = H_2(\mathbf{w^j}0, R_{\mathbf{w^j}0})$ and $h_{\mathbf{w^j}1} = H_2(\mathbf{w^j}1, R_{\mathbf{w^j}1})$. Push $(S_{\mathbf{w^j}1}, R_{\mathbf{w^j}1})$ and $(S_{\mathbf{w^j}0}, R_{\mathbf{w^j}0})$ onto the stack orderly. Let X_{j+1} denote the current stack and define $\Omega_{j+1} = \Omega_j \bigcup \{R_{\mathbf{w^j}0}\}$.

b) If $\mathbf{w^j}$ is a leaf, define X_{j+1} with the current stack.

i) If $w_t = 0$ (Note that the node $\mathbf{w^{j+1}}$ is the right sibling node of $\mathbf{w^j}$ in this case), then set $\Omega_{j+1} = \Omega_j \bigcup \{R_{\mathbf{w^{j+1}}}\} - \{R_{\mathbf{w^j}}\}$ ($R_{\mathbf{w^{j+1}}}$ can be read from the new top $(S_{\mathbf{w^{j+1}}}, R_{\mathbf{w^{j+1}}})$ of the stack).

ii) If $w_t = 1$ (Note that $\mathbf{w^{j+1}} = w''1$ in this case, where w'' is the longest string such that $w''0$ is a prefix of $\mathbf{w^j}$), then set $\Omega_{j+1} = \Omega_j \bigcup \{R_{\mathbf{w^{j+1}}}\} - \{R_{w''0}, R_{w''01}, \cdots, R_{w_t}\}$.

c) Finally, return $SK_{j+1} = (X_{j+1}, \Omega_{j+1})$.

We give an example to show the stack changes from time period 0 to time period 9 when $l = 4$ in Fig.4. As shown is Fig. 3, the time periods j(j=3,4,6,7) correspond to the leaves of the binary tree in this example. So the $KeyUpdate$ algorithm should run b and c steps in these time periods. While other time periods j(j=0,1,2,5,8,9) correspond to the internal nodes of the binary tree. So the $KeyUpdate$ algorithm should run a and c steps in these time periods. The changes of $\Omega_j (0 \le j \le 9)$ are shown as follows.

$\Omega_0 = \varnothing$,
$\Omega_1 = \Omega_0 \cup \{R_0\}$=$\{R_0\}$,
$\Omega_2 = \Omega_1 \cup \{R_{00}\}$=$\{R_0, R_{00}\}$,
$\Omega_3 = \Omega_2 \cup \{R_{000}\}$=$\{R_0, R_{00}, R_{000}\}$
$\Omega_4 = \Omega_3 \cup \{R_{001}\} - \{R_{000}\}$=$\{R_0, R_{00}, R_{001}\}$,
$\Omega_5 = \Omega_4 \cup \{R_{01}\} - \{R_{00}, R_{001}\}$=$\{R_0, R_{01}\}$,
$\Omega_6 = \Omega_5 \cup \{R_{010}\} = \{R_0, R_{01}, R_{010}\}$,
$\Omega_7 = \Omega_6 \cup \{R_{011}\} - \{R_{010}\} = \{R_0, R_{01}, R_{011}\}$,
$\Omega_8 = \Omega_7 \cup \{R_1\} - \{R_0, R_{01}, R_{011}\} = \{R_1\}$,
$\Omega_9 = \Omega_8 \cup \{R_{10}\} = \{R_1, R_{10}\}$.

3) $AuthGen$: Input the public key PK, the current period j, a client's secret key SK_j, and a file $F = \{m_1, \cdots, m_n\}$, where $m_i \in Z_q^* (i = 1, \cdots, n)$. Denote the node associated with period j with a binary string $\mathbf{w^j} = w_1 \cdots w_t$.

a) The client parses $SK_j = (X_j, \Omega_j)$ and reads the top element $(S_{\mathbf{w^j}}, R_{\mathbf{w^j}})$ from the stack X_j. The client selects $r \in Z_q^*$ randomly, and computes $U = g^r$ and $\sigma_i = H_3(name||i||j, U)^r \cdot S_{\mathbf{w^j}} \cdot u^{rm_i}$ for blocks $m_i (i = 1, \cdots, n)$, where the name $name$ is chosen by the client uniformly at random from Z_q^* as the identifier of file F. In order to ensure the integrity of unique file identifier $name$, time period j and the set Ω_j (they can be viewed as the public values used for verification that do not change with actual files), the client also generates a file tag for $name$, j and Ω_j using $SSig$.

b) Denote the set of authenticators in time period j with $\Phi = (j, U, \{\sigma_i\}_{1 \le i \le n}, \Omega_j)$.

c) Finally, the client sends the file F and the set of authenticators along with the file tag to the cloud.

4) $ProofGen$: Input the public key PK, a time period j, a challenge $Chal = \{(i, v_i)\}_{i \in I}$ (where $I = \{s_1, \cdots, s_c\}$ is a c-element subset of set $[1, n]$ and $v_i \in Z_q^*$), a file F and the set of authenticators $\Phi = (j, U, \{\sigma_i\}_{1 \le i \le n}, \Omega_j)$. Here, $(j, Chal)$ pair is issued by the auditor, and then used by the cloud.

The cloud calculates an aggregated authenticator $\Phi = (j, U, \sigma, \Omega_j)$, where $\sigma = \prod_{i \in I} \sigma_i^{v_i}$. It also computes the linear combination of sampled blocks $\mu = \sum_{i \in I} v_i m_i$,where $m_i \in Z_q^*$. It then sends $P = (j, U, \sigma, \mu, \Omega_j)$ along with the file tag as the response proof of storage correctness to the client.

5) $ProofVerify$: Input the public key PK, a time period j, a challenge $Chal$ and a proof P. Denote the node associated with period j with a binary string $\mathbf{w^j} = w_1 \cdots w_t$.

The client parses $\Omega_j = (R_{\mathbf{w^j}|_1}, \cdots, R_{\mathbf{w^j}|_t})$. He then verifies the integrity of $name$, j and Ω_j by checking the file tag. After that, the client verifies whether the

TABLE II
EFFICIENCY COMPARISON

Protocols	SysSetup	KeyUpdate	AuthGen	ProofGen	ProofVerify
Our Protocol	$2T_e$	$4T_e$	$3T_e$	cT_e	$3T_p + (log(T+2)+c+1)T_e$
Shacham *et al.*'s protocol [5]	T_e		$2T_e$	cT_e	$2T_p + cT_e$

TABLE III
COMPLEXITIES OF KEYS SIZE AND COMMUNICATION OVERHEADS

Protocols	Secret key size	Public key size	Challenge overhead	Response overhead
Our Protocol	$O((logT)k)$	$O(k)$	$O(k)$	$O((logT)k)$
Shacham *et al.*'s protocol [5]	$O(k)$	$O(k)$	$O(k)$	$O(k)$

following equation holds:

$$\hat{e}(R \cdot \prod_{m=1}^{t} R_{\mathbf{w^j}|_m}^{h_{\mathbf{w^j}|_m}}, H_1(R)^{\sum_{i\in I} v_i}) \cdot \hat{e}(U, u^{\mu} \cdot \prod_{i\in I} H_3(name||i||j,U)^{v_i}) = \hat{e}(g,\sigma),$$

where $h_{\mathbf{w^j}} = H_2(\mathbf{w^j}, R_{\mathbf{w^j}})$

If it holds, returns "$True$", otherwise returns "$False$".

IV. SECURITY AND PERFORMANCE

A. Security Analysis

Theorem 1 (Correctness): For each random challenge $Chal$ and one valid proof P, the ProofVerify algorithm always returns "$True$".

Proof. It is because the following equations hold:

$$\begin{aligned}
&\hat{e}(R \cdot \prod_{m=1}^{t} R_{\mathbf{w^j}|_m}{}^{h_{\mathbf{w^j}|_m}}, H_1(R)^{\sum_{i\in I} v_i}) \cdot \hat{e}(U, u^{\mu} \cdot \prod_{i\in I} H_3(name||i||j,U)^{v_i})\\
&= \hat{e}(\prod_{i\in I} g^{v_i(\rho+\sum_{m=1}^{t}\rho_{\mathbf{w^j}|_m} h_{\mathbf{w^j}|_m})}, H_1(R)) \cdot \hat{e}(g, u^{r\mu}) \cdot \hat{e}(U, \prod_{i\in I} H_3(name||i||j,U)^{v_i})\\
&= \hat{e}(g, \prod_{i\in I} H_1(R)^{v_i(\rho+\sum_{m=1}^{t}\rho_{\mathbf{w^j}|_m} h_{\mathbf{w^j}|_m})}) \cdot \hat{e}(g, u^{\sum_{i\in I} rm_i v_i}) \cdot \hat{e}(g^r, \prod_{i\in I} H_3(name||i||j,U)^{v_i})\\
&= \hat{e}(g, \prod_{i\in I} H_3(name||i||j,U)^{v_i r}) \cdot \hat{e}(g, \prod_{i\in I} (H_1(R)^{v_i(\rho+\sum_{m=1}^{t}\rho_{\mathbf{w^j}|_m} h_{\mathbf{w^j}|_m})} \cdot u^{rm_i v_i}))\\
&= \hat{e}(g, \prod_{i\in I} (H_3(name||i||j,U)^r \cdot H_1(R)^{\rho+\sum_{m=1}^{t}\rho_{\mathbf{w^j}|_m} h_{\mathbf{w^j}|_m}} \cdot u^{rm_i})^{v_i})\\
&= \hat{e}(g, \prod_{i\in I} \sigma_i{}^{v_i})\\
&= \hat{e}(g, \sigma)
\end{aligned}$$

Theorem 2 (Key-exposure resistance): If the CDH problem in G_1 is hard and the signature scheme $SSig$ used for file tags is existentially unforgeable, then our proposed auditing protocol is key exposure resistant.

Proof. Our proof is mainly composed by two phases. The first phase is to prove the verifier will reject except when the prover responds with correctly computed values in $P = (j, U, \sigma, \mu, \Omega_j)$. This proof idea is similar to that in [5]. The second phase is to construct a knowledge extractor to extract the whole challenged file blocks. The proof idea is similar to [1]. Please see Appendix for the detailed proof.

Theorem 3 (Detectability): Our auditing protocol is $(\frac{t}{n}, 1-(\frac{n-1}{n})^c)$ detectable if the cloud stores a file with n blocks, and deletes or modifies t blocks.

The proof of Theorem 3 is easy. According to our previous definitions, the cloud stores a file with total n blocks including t bad (deleted or modified) blocks. The number of challenged blocks is c. Thus, the bad blocks can be detected if and only if at least one of the challenged blocks picked by the auditor matches the bad blocks. The probability analysis is the same as the probability analysis of deletion detection in [1] and the probability analysis of Theorem 4 in [14]. We omit it here.

B. Performance Analysis

In Table 2, we give the efficiency comparison between our protocol and Shacham *et al.*'s protocol based on BLS signature [5]. We choose Shacham *et al.*'s protocol [5] as a benchmark is mainly because the construction of it is generally viewed as very efficient. It is also the most related to our construction. Here, T_e denotes the time costs of exponentiation on the group G_1, and T_p denotes the time costs of bilinear pairing from G_1 to G_2. Other operations like the multiplication on G_1, the operations on Z_q and G_2, set operations, stack operations and hashing operations are omitted because they just contribute negligible computation costs. Note that it is natural for our protocol to add more overhead than Shacham *et al.*'s protocol in order to achieve the extra key-exposure resilience. Because we employ the binary tree structure and the pre-order traversal technique to associate the time periods and update secret keys, as shown in Table 2, our protocol achieves nice performance. The costs of the $SysSetup$ algorithm, the $KeyUpdate$ algorithm, the $AuthGen$ algorithm, the $ProofGen$ algorithm are independent of the total number of time periods T. The cost of the $ProofVerify$ algorithm is only logarithmic in T. What's important is that our $ProofVerify$ algorithm only requires three (independent of T) pairing operations, while pairing is well considered as very time-consuming among many cryptographic atomic operations. Thus the running time of the $ProofVerify$ algorithm in our protocol is dominated by $3T_p$ as long as c and T are not extremely large. Therefore, our protocol only adds reasonable computation overhead compared with Shacham *et al.*'s protocol [5].

In Table 3, we give the complexity comparison of keys size and communication overhead between our protocol and Shacham *et al.*'s protocol [5]. The secret key size of the client is only logarithmic in T and the public key size is independent of T. It is much better than the slightly better solution in Section 3, in which the public key size and the secret key size are both linear with T. The complexities of the challenge overhead in our protocol and Shacham *et al.*'s protocol [5] are both $O(k)$ (here k is the security parameter). The complexity of the response overhead is $O((logT)k)$ in our protocol because the response proof needs to contain the set of verification values X_j. Generally speaking, the complexities of communication overhead and key size in our protocol are at most logarithmic in T, which is completely acceptable in practice.

V. Discussions

A. Support the TPA

Our proposed protocol can easily be modified to support the TPA because we have considered the public verification during our design. In the modified auditing protocol supporting the TPA, the $SysSetup$ algorithm, the $KeyUpdate$ algorithm and the $AuthGen$ algorithm are the same as the description in Section 3. In the $ProofGen$ algorithm, we only modify our original protocol as follows: The TPA generates a challenge $Chal = \{(i, v_i)\}_{i \in I}$, and sends it to the cloud. After the cloud completes the same operations as those in original protocol in Section 3, it sends the proof P to the TPA instead of the client. In the $ProofVerify$ algorithm, we only need to make the TPA instead of the client verify the validity of the tag and the proof P.

B. Support lazy update

Sometime, a client prefers to update his secret key after multiple time periods. It is called lazy update. For example, the client may be very busy from time period i to time period $j(j > i)$ and has no time to update his secret keys during these periods. Our protocol supports lazy update. The client can update his secret key only at the end of time period j. In fact, the computation of lazy update may be much less than that of executing the $KeyUpdate$ algorithm several times. It is because some operations in the $KeyUpdate$ algorithm may be omitted. Let us give an example to show how the lazy update works. In Fig. 3, if a client would like to directly update his secret key from time period 1 to time period 5, he only needs to compute R_{01} and S_{01}. Note that R_{00}, R_{000}, R_{001}, S_{00}, S_{000} and S_{001} do not need to be computed in this case.

However, lazy update may increase the probability of key exposure because the secret keys have not been updated by a lazy client for a longer time. Therefore, we suggest it is only used as an option of special needs for clients, as a tradeoff between efficiency and security.

C. Support multiple sectors

As described in [9], we can divide one block into s sectors in our auditing protocol. In this case, the sector size instead of the block size is restricted by the security parameter. By doing this, we can generate one authenticator for each block that contains s sectors. The total number of authenticators can become smaller, and the whole storage overhead, introduced by authenticators, can be reduced [9]. The description of our auditing protocol can be viewed as an instance chosen $s = 1$.

D. Support blockless verification

The blockless verifiability means that the cloud can construct a proof that allows the auditor to check the integrity of certain file blocks in cloud, even when the auditor does not have access to the actual file blocks [1]. In the $ProofVerify$ algorithm of our protocol, the auditor can check the integrity of certain file blocks without having access the actual file blocks $m_i(i \in I)$. The auditor only uses the proof $P = (j, U, \sigma, \mu, \Omega_j)$ generated by cloud to check the integrity of file blocks. So our protocol supports blockless verification.

VI. Related work

In order to check the integrity of the data stored in the remote server, many protocols were proposed [1–12, 14, 30]. These protocols focused on various requirements such as high efficiency, stateless verification, data dynamic operation, privacy protection, etc. According to the role of the auditor, these auditing protocols can be divided into two categories: private verification and public verification. In an auditing protocol with private verifiability, the auditor is provided with a secret that is not known to the prover or other parties. Only the auditor can verify the integrity of the data. In contrast, the verification algorithm does not need a secret key from the auditor in an auditing protocol with public verifiability. Therefore, any third party can play the role of the auditor in this kind of auditing protocols.

Ateniese *et al.* [1] firstly considered the public verification and proposed the notion of "provable data possession" (PDP) for ensuring data possession at untrusted storages. They used the technique of HLA and random sample to audit outsourced data. Juels and Kaliski Jr. [30] explored a "proof of retrievability" (PoR) model. They used the tools of spot-checking and error-correcting codes to ensure both possession and retrievability of files on remote storage systems. Shacham and Waters [5] gave two short and efficient homomorphic authenticators: one has private verifiability which is based on pseudorandom functions; the other has public verifiability which is based on the BLS signature. Dodis *et al.* [31] focused on the study on different variants of existing POR work. Shah *et al.* [32] introduced a TPA to keep online storage honest. The protocol requires the auditor to maintain the state, and suffers from bounded usage. Wang *et al.* [10] provided a public auditing protocol that has privacy-preserving property. In order to make the protocol achieve privacy-preserving property, they integrate the HLA with random masking technique. Wang proposed a proxy provable data possession protocol [14]. In this protocol, the client delegates its data integrity checking task to a proxy. Dynamic data operations for audit services are also attended in order to make auditing more flexible. Ateniese *et al.* [2] firstly proposed a partially dynamic PDP protocol.

Wang *et al*. [11] proposed another auditing protocol supporting data dynamics. In this protocol, they utilized the BLS-based HLA and Merkle Hash Tree to support fully data dynamics. Erway *et al*. [13] extended the PDP mode and proposed a skip list-based protocol with dynamics support. In [33], Zhu *et al*. proposed a cooperative provable data possession protocol which can be extended to support the dynamic auditing. Yang and Jia [9] proposed a dynamic auditing protocol with privacy-preserving property. The problem of user revocation in cloud storage auditing was considered in [15]. Recently, some dynamic PoR approaches [34–36] and provable outsourced version control systems [37, 38] have been studied.

Most of above auditing protocols are all built on the assumption that the secret key of the client is absolutely secure and would not be exposed. But as we have shown previously, this assumption may not always be true. Our work advances the field by exploring how to achieve key-exposure resistance in cloud storage auditing, under the new problem settings.

VII. Conclusion

In this paper, we study on how to deal with the client's key exposure in cloud storage auditing. We propose a new paradigm called auditing protocol with key-exposure resilience. In such a protocol, the integrity of the data previously stored in cloud can still be verified even if the client's current secret key for cloud storage auditing is exposed. We formalize the definition and the security model of auditing protocol with key-exposure resilience, and then propose the first practical solution. The security proof and the asymptotic performance evaluation show that the proposed protocol is secure and efficient.

Appendix A
Proof of the Theorem 2

Proof. Similarly to the idea in [5], we will define a series of games, and give the analysis limiting the difference in adversary behavior between successive games.

Game 0. Game 0 is simply the game defined in Table 1.

Game 1. Game 1 is the same as Game 0, with only one difference. The challenger stores a list of all signed tags issued as part of authenticator queries. If the adversary submits one tag (in initiating the auditing protocol or as the challenge tag), the challenger will abort when this tag is a valid signature generated through $SSig$ signature algorithm but not signed by the challenger.

Analysis. This is very similar to the analysis in [5]. Obviously, if the adversary can make the challenger abort in Game 1 with non-negligible probability, it is easy to use the adversary to construct a forger against the $SSig$ signature scheme. From now on, we can assure that $name$, j and any verification value in $\Omega_j = (R_{\mathbf{w^j}|_1}, \cdots, R_{\mathbf{w^j}|_t})$ used in the interactions with the adversary are generated by the challenger.

Game 2. Game 2 is the same as Game 1, with only one difference. The challenger stores a list of his responses to authenticator queries from the adversary. If in the forgery phase the adversary wins the game but U in its proof P is different from the real U in the set of authenticators $\Phi = (j, U, \{\sigma_i\}_{1\leq i\leq n}, \Omega_j)$ that the challenger has stored, then the challenger will abort.

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

Firstly, the simulator is given a tuple $(g, I_1 = H_1(R) = g^{a'} \in G_1, I_2 = R = g^{b'} \in G_1)$. The aim of the simulator is to compute $g^{a'b'}$, where $\rho = b', a' \in Z_q^*$ are unknown to the simulator and adversary $\mathcal{A}$. Hash function H_3 is viewed as a random oracle. As the same method as [24, 26, 27], the hash function H_2 will be replaced by one $(l+1)$-wise independent hash function in function family $\mathcal{H}_2$. Let $\mathbf{w^b} = w_1^* w_2^* \cdots w_s^*$ be the bit string of the node corresponding to the break-in time period b.

(1)Setup phase. The simulator selects a total time periods T. He randomly guesses a time period b in which $\mathcal{A}$ enters the break-in phase. Obviously, the probability that his guess is correct is $1/T$(non-negligible). The simulator randomly selects $\alpha \in Z_q^*$, and sets $u = g^\alpha$. He randomly selects $h_{w_1^* w_2^* \cdots w_s^*}$, $y_{w_1^* w_2^* \cdots w_s^*}$, $h_{w_1^* w_2^* \cdots w_{\eta-1}^* 1}$ and $y_{w_1^* w_2^* \cdots w_{\eta-1}^* 1}$ in Z_q^*, where $1 \leq \eta \leq s$ and $w_\eta^* = 0$. He also samples at random a hash function H_2 from one $(l+1)$-wise independent hash function family $\mathcal{H}_2$ with the following constraints for $1 \leq \eta \leq s$ and $w_\eta^* = 0$.

$$H_2(w_1^* w_2^* \cdots w_s^*, R_{w_1^* w_2^* \cdots w_s^*}) = h_{w_1^* w_2^* \cdots w_s^*},$$

where $R_{w_1^* w_2^* \cdots w_s^*} = (g^{y_{w_1^* w_2^* \cdots w_s^*}}/R)^{1/h_{w_1^* w_2^* \cdots w_s^*}}$.

$$H_2(w_1^* w_2^* \cdots w_{\eta-1}^* 1, R_{w_1^* w_2^* \cdots w_{\eta-1}^* 1}) = h_{w_1^* w_2^* \cdots w_{\eta-1}^* 1},$$

where $R_{w_1^* w_2^* \cdots w_{\eta-1}^* 1} = (g^{y_{w_1^* w_2^* \cdots w_{\eta-1}^* 1}}/R)^{1/h_{w_1^* w_2^* \cdots w_{\eta-1}^* 1}}$.

The simulator provides $PK = (G_1, G_2, \hat{e}, g, u, T, R)$ to $\mathcal{A}$.

(2)Query phase. In order to answer the query of $\mathcal{A}$, the simulator needs to update keys as his preparation. Let $\mathbf{w^j} = w_1 \cdots w_t$ denote the node corresponding to the current time period $j(j < b)$. He simulates the key update algorithm as follows.

(1) if $\mathbf{w^j}$ is the leaf, the simulator does not need to update keys.

(2) If $\mathbf{w^j}0 = \mathbf{w^b}$, then the simulator has defined $R_{\mathbf{w^j}0} = (g^{y_{\mathbf{w^j}0}}/R)^{1/h_{\mathbf{w^j}0}}$, $R_{\mathbf{w^j}1} = (g^{y_{\mathbf{w^j}1}}/R)^{1/h_{\mathbf{w^j}1}}$, $H_2(\mathbf{w^j}0, R_{\mathbf{w^j}0}) = h_{\mathbf{w^j}0}$ and $H_2(\mathbf{w^j}1, R_{\mathbf{w^j}1}) = h_{\mathbf{w^j}1}$ for some $y_{\mathbf{w^j}0}, y_{\mathbf{w^j}1}, h_{\mathbf{w^j}0}, h_{\mathbf{w^j}1} \in Z_q^*$ during choosing H_2 from hash function family $\mathcal{H}_2$.

(3) If $\mathbf{w^j}0 \neq \mathbf{w^b}$ is a prefix of $\mathbf{w^b}$, then the simulator randomly selects $\rho_{\mathbf{w^j}0}, h_{\mathbf{w^j}0} \in Z_q^*$, and sets $R_{\mathbf{w^j}0} = g^{h_{\mathbf{w^j}0}}$ and $H_2(\mathbf{w^j}0, R_{\mathbf{w^j}0}) = h_{\mathbf{w^j}0}$. The simulator has defined $R_{\mathbf{w^j}1} = (g^{y_{\mathbf{w^j}1}}/R)^{1/h_{\mathbf{w^j}1}}$ and $H_2(\mathbf{w^j}1, R_{\mathbf{w^j}1}) = h_{\mathbf{w^j}1}$ for some $y_{\mathbf{w^j}1}, h_{\mathbf{w^j}1} \in Z_q^*$ during choosing H_2 from hash function family $\mathcal{H}_2$.

(4) Otherwise, the simulator randomly selects $\rho_{\mathbf{w^j}0}, \rho_{\mathbf{w^j}1}, h_{\mathbf{w^j}0}, h_{\mathbf{w^j}1} \in Z_q^*$, and computes $R_{\mathbf{w^j}0} = g^{h_{\mathbf{w^j}0}}$, $R_{\mathbf{w^j}1} = g^{h_{\mathbf{w^j}1}}$, $H_2(\mathbf{w^j}0, R_{\mathbf{w^j}0}) = h_{\mathbf{w^j}0}$ and $H_2(\mathbf{w^j}1, R_{\mathbf{w^j}1}) = h_{\mathbf{w^j}1}$.

H_3 queries. The simulator keeps a list of queries. When $\mathcal{A}$ queries H_3 oracle at a point $< name||i||j, U >$, the simulator does as follows.

He checks whether $< name||i||j, U >$ is in a tuple $< name||i||j, U, h, \lambda, \gamma >$ of H_3 table.

(1)If it is, the simulator returns h to $\mathcal{A}$.

(2)Otherwise, the simulator randomly selects $\lambda \in Z_q^*$, and computes $h = g^{\lambda}$. He adds tuple $< name||i||j, U, h, \lambda, * >$ to H_3 table and returns h to $\mathcal{A}$.

Authenticator Queries. The simulator keeps a list of queries. When $\mathcal{A}$ queries the authenticator of any block m_i with name *name* in time period j, the simulator does as follows.

(1) He randomly selects $\lambda_i, \gamma \in Z_q^*$, and computes $U = R^{\gamma}$ and $h = g^{\lambda_i} \cdot I_1^{-1/\gamma}$. If $H_3(name||i||j, U)$ has been defined, then aborts. Because the space from which names are selected is very large [5] and U is random, except with negligible probability, the same $name$ and U have been chosen by the simulator before for some other file and a query has not been made to this random oracle at $< name||i||j, U >$ for some i and j. The simulator adds $< name||i||j, U, h, \lambda_i, \gamma >$ to H_3 table.

(2) He computes $\psi = R^{\lambda_i\gamma+\alpha\gamma m_i} \cdot I_1^{\sum_{m=1}^{t} h_{\mathbf{wj}|_m}\rho_{\mathbf{wj}|_m}}$ and sets $\Omega_j = (R_{\mathbf{wj}|_1}, \cdots, R_{\mathbf{wj}|_t})$(Note that all $R_{\mathbf{wj}|_m}(1 \leq m \leq t)$ have been generated by the simulator). And then he sends the authenticator (j, U, ψ, Ω_j) to $\mathcal{A}$. Note that $h_{\mathbf{wj}|_m}, \rho_{\mathbf{wj}|_m}(1 \leq m \leq t)$ have been defined during the key update simulation and the following equations hold.

$$\begin{aligned}\psi &= H_3(name||i||j, U)^{\rho\gamma} \cdot {I_1}^{\rho+\sum_{m=1}^{t} h_{\mathbf{wj}|_m}\rho_{\mathbf{wj}|_m}} \cdot u^{\rho\gamma m_i}\\ &= (g^{\lambda_i} \cdot {I_1}^{-1/\gamma})^{\rho\gamma} \cdot {I_1}^{\rho+\sum_{m=1}^{t} h_{\mathbf{wj}|_m}\rho_{\mathbf{wj}|_m}} \cdot g^{\alpha\rho\gamma m_i}\\ &= R^{\lambda_i\gamma+\alpha\gamma m_i} \cdot {I_1}^{\sum_{m=1}^{t} h_{\mathbf{wj}|_m}\rho_{\mathbf{wj}|_m}}\end{aligned}$$

(3)Break-in Phase. In this phase, the simulator needs to provide the current secret key to $\mathcal{A}$. From the simulated key update algorithm, the simulator has known the verification values $R_{\mathbf{w^b}|_{\bar\eta}}(1 \leq \eta \leq s)$ and can generate the node secret keys $S_{\mathbf{w^b}|_{\bar\eta}} = I_1^{y_{\mathbf{w^b}|_{\bar\eta}}+\sum_{m=1}^{\eta-1} h_{\mathbf{w^b}|_m}\rho_{\mathbf{w^b}|_m}}(1 \leq \eta \leq s)$ for the right siblings of the nodes $\mathbf{w^b}|_\eta$ (if they have right siblings) on the path from the root to $\mathbf{w^b}$ using $h_{\mathbf{w^b}|_m}$, $\rho_{\mathbf{w^b}|_m}$ for $1 \leq m \leq \eta - 1$ and $y_{\mathbf{w^b}|_{\bar\eta}}$. It is because

$$\begin{aligned}S_{w^b|_{\bar\eta}} &= {I_1}^{\rho+\sum_{m=1}^{\eta-1} \rho_{w^b|_m}h_{w^b|_m}+\rho_{w^b|_{\bar\eta}}h_{w^b|_{\bar\eta}}}\\ &= {I_1}^{\rho+\sum_{m=1}^{\eta-1} \rho_{w^b|_m}h_{w^b|_m}+(1/h_{w^b|_{\bar\eta}})(y_{w^b|_{\bar\eta}}-\rho)h_{w^b|_{\bar\eta}}}\\ &= {I_1}^{y_{w^b|_{\bar\eta}}+\sum_{m=1}^{\eta-1} h_{w^b|_m}\rho_{w^b|_m}}\end{aligned}$$

Similarly, the simulator has known the verification value $R_{\mathbf{w^b}}$ and can generate $S_{\mathbf{w^b}}$ according to $h_{w^b|_m}, \rho_{w^b|_m}(1 \leq m \leq s-1)$ and y_{w^b} generated during the key update simulation. It is because

$$\begin{aligned}S_{w^b} &= {I_1}^{\rho+\sum_{m=1}^{s} \rho_{w^b|_m}h_{w^b|_m}}\\ &= {I_1}^{\rho+\sum_{m=1}^{s-1} \rho_{w^b|_m}h_{w^b|_m}+(1/h_{w^b})(y_{w^b}-\rho)h_{w^b}}\\ &= {I_1}^{y_{w^b}+\sum_{m=1}^{s-1} h_{w^b|_m}\rho_{w^b|_m}}\end{aligned}$$

From the simulated key update algorithm, the simulator has known $\Omega_b = (R_{w^b|_1}, \cdots, R_{w^b|_s})$. So the simulator can return $SK_b = (X_b, \Omega_b)$ to the adversary $\mathcal{A}$.

(4)Challenge Phase

The simulator randomly selects a time period j^* and a challenge $Chal = \{(i, v_i)\}_{i\in I}$, where $I = \{s_1, s_2, \cdots, s_c\}$, and provides the adversary with $Chal$. It requests the adversary to provide a proof of possession of file $F = \{m_1, \cdots, m_n\}$ under $Chal$ for the blocks $\{m_{s_1}, \cdots, m_{s_c}\}$ in time period j^*, where $1 \leq s_l \leq n$, $1 \leq l \leq c$, $1 \leq c \leq n$.

(5)Forgery phase. The adversary outputs a proof of possession P for the blocks indicated by $Chal$ and returns $P = (j, U, \sigma, \mu, \Omega_j)$ as the response proof of storage correctness to the client. If ProofVerify$(j^*, PK, Chal, P) =$ "$True$", then

$$\hat{e}(R \cdot \prod_{m=1}^{t} R_{w^{j^*}|_m}^{h_{w^{j^*}|_m}}, H_1(R)^{\sum_{i\in I} v_i}) \cdot \hat{e}(U, u^{\mu} \cdot \prod_{i\in I} H_3(name||i||j^*, U)^{v_i}) = \hat{e}(g, \sigma).$$

If U in the adversary's proof $P = (j, U, \sigma, \mu, \Omega_j)$ is different from the real U in the set of authenticators $\Phi = (j, U, \{\sigma_i\}_{1\leq i\leq n}, \Omega_j)$ that the simulator has stored (Note that Φ is got from authenticator queries), the simulator can abstract tuple $< name||i||j^*, U, h, \lambda_i, * >$ from H_3 table to get $H_3(name||i||j^*, U) = g^{\lambda_i}$ except possibly with negligible probability. Note that the probability $\sum_{i\in I} v_i = 0$ is negligible. Using the $h_{w^{j^*}|_m}$, $\rho_{w^{j^*}|_m}$ generated in the key update simulation, the simulator can solve the CDH problem.

$$\begin{aligned}&\hat{e}(R \cdot \prod_{m=1}^{t} {R_{w^{j^*}|_m}}^{h_{w^{j^*}|_m}}, H_1(R)^{\sum_{i\in I} v_i}) \cdot\\ &\hat{e}(U, u^{\mu} \cdot \prod_{i\in I} H_3(name, i, j^*, U)^{v_i}) = \hat{e}(g, \sigma) \qquad \Rightarrow\\ &\hat{e}(R^{\sum_{i\in I} v_i} \cdot \prod_{m=1}^{t} g^{\rho_{w^{j^*}|_m}h_{w^{j^*}|_m}\cdot\sum_{i\in I} v_i}, H_1(R)) \cdot\\ &\hat{e}(U, g^{\alpha\mu} \cdot \prod_{i\in I} g^{\lambda_i v_i}) = \hat{e}(g, \sigma) \qquad \Rightarrow\\ &\hat{e}(g^{b'\cdot\sum_{i\in I} v_i}, g^{a'}) \cdot \hat{e}(g^{\sum_{m=1}^{t} \rho_{w^{j^*}|_m}h_{w^{j^*}|_m}\cdot\sum_{i\in I} v_i}, I_1) \cdot\\ &\hat{e}(U, g^{\alpha\mu+\sum_{i\in I} \lambda_i v_i}) = \hat{e}(g, \sigma) \qquad \Rightarrow\\ &g^{a'b'} = (\sigma \cdot U^{-(\alpha\mu+\sum_{i\in I} \lambda_i v_i)} \cdot\\ &\qquad {I_1}^{-\sum_{m=1}^{t} \rho_{w^{j^*}|_m}h_{w^{j^*}|_m}\cdot\sum_{i\in I} v_i})^{1/\sum_{i\in I} v_i}\end{aligned}$$

So U in prover's proof $P = (j, U, \sigma, \mu, \Omega_j)$ should be correct. It means that the difference between the adversary's probability of success in Game 1 and 2 is negligible.

Game 3. Game 3 is the same as Game 2, with only one difference. The challenger stores a list of its responses to authenticator queries from the adversary. The challenger observes each instance of the cloud auditing protocol with the adversary. If in any instance the adversary is successful but the adversary's aggregate authenticator σ is not equal to $\sigma = \prod_{i\in I} \sigma_i^{v_i}$, then the challenger will abort.

Analysis. Suppose the file that causes the abort in time period j has name $name$, is composed by $m_1, \cdots, m_n$, and that the set of authenticators issued by challenger are $\Phi = (j, U, \{\sigma_i\}_{1\leq i\leq n}, \Omega_j = (R_{\mathbf{wj}|_1}, \cdots, R_{\mathbf{wj}|_t}))$. Let $(j, Chal = \{(i, v_i)\}_{i\in I})$ be the query that makes the challenger abort, and $P = (j, U, \sigma', \mu', \Omega_j)$ be the proof of the adversary. Let the expected response that would have been got from an honest prover be $P = (j, U, \sigma, \mu, \Omega_j)$. So the correctness of the proof can be verified by the following equation

$$\hat{e}(R \cdot \prod_{m=1}^{t} R_{\mathbf{wj}|_m}^{h_{\mathbf{wj}|_m}}, H_1(R)^{\sum_{i\in I} v_i}) \cdot \hat{e}(U, u^{\mu} \cdot \prod_{i\in I} H_3(name||i||j, U)^{v_i}) = \hat{e}(g, \sigma).$$

Because the challenger aborts, we can know that $\sigma \neq \sigma'$ and σ' can pass the verification equation, i.e., that

$$\hat{e}(R \cdot \prod_{m=1}^{t} R_{\mathbf{wj}|_m}^{h_{\mathbf{wj}|_m}}, H_1(R)^{\sum_{i\in I} v_i}) \cdot \hat{e}(U, u^{\mu'} \cdot \prod_{i\in I} H_3(name||i||j,U)^{v_i}) = \hat{e}(g, \sigma').$$

Obviously, $\mu \neq \mu'$, otherwise, it means $\sigma = \sigma'$, which contradicts the assumption above. Let $\Delta\mu = \mu' - \mu$.

We now construct a simulator to solve the CDH problem if the adversary can make the challenger abort with non-negligible probability.

The simulator is given a tuple $(g, g^{a'}, v)$; his goal is to compute $v^{a'}$. The simulator acts like the challenger in Game 2, with the following differences:

In Setup phase, the simulator selects $\beta, \gamma \in_R Z_q^*$ and sets $u = g^\beta v^\gamma$.

The simulator controls a random oracle H_3, and stores a list of random oracle queries. When the adversary queries H_3 oracle at a point $< name||i||j, U >$, the simulator checks whether $< name||i||j, U >$ is in a tuple $< name||i||j, U, h, \lambda, \gamma >$ of H_3 table. If it is, the challenger returns h to the adversary; Otherwise, the simulator randomly selects $\lambda \in Z_q^*$, and computes $h = g^\lambda$. It adds tuple $< name||i||j, U, h, \lambda, * >$ to H_3 table and returns h to the adversary.

The simulator stores a list of authenticator queries. When the adversary queries the authenticator of any block m_i with name *name* in time period j, the simulator randomly selects $\lambda_i, \eta \in Z_q^*$, computes $U = (g^{a'})^\eta$ and sets $h = g^{\lambda_i}/(g^\beta v^\gamma)^{m_i}$. As we have analysed before, the probability that $H_3(name||i||j,U)$ has been defined is negligible. The simulator can compute authenticator because $\sigma_i = H_3(name||i||j,U)^{a'\eta} \cdot S_{\mathbf{wj}} \cdot u^{a'\eta m_i} = (g^{\lambda_i}/(g^\beta v^\gamma)^{m_i})^{a'\eta} \cdot S_{\mathbf{wj}} \cdot ((g^\beta v^\gamma)^{m_i})^{a'\eta} = (g^{a'})^{\lambda_i \eta} \cdot S_{\mathbf{wj}}$. Note that the simulator knows $S_{\mathbf{wj}}$ because he selects SK_0 by himself and can generate secret keys of all the time periods. The challenger adds $< name||i||j, U, h, \lambda_i, \eta >$ to H_3 table.

In break in phase, the simulator generates SK_b by the $KeyUpdate$ algorithm and sends it to the adversary.

If the adversary succeeds in the game in responding with a proof $P = (j, U, \sigma', \mu', \Omega_j)$ in which σ' is different from the expected σ, we can extract an tuple $< name||i||j, U, h, \lambda_i, \eta >$ ($i \in I$) from H_3 table (it must have been generated when the adversary makes an authenticator query). And then we can divide the verification equation for forged σ' by the verification equation for the expected σ to get $\hat{e}(\sigma'/\sigma, g) = \hat{e}(U, u^{\Delta\mu}) = \hat{e}(U, (g^\beta v^\gamma)^{\Delta\mu})$. So $\hat{e}(\sigma' \cdot \sigma^{-1} \cdot U^{-\beta\Delta\mu}, g) = \hat{e}((g^{a'})^\eta, v)^{\gamma\Delta\mu}$. We can solve the CDH problem $v^{a'} = (\sigma' \cdot \sigma^{-1} \cdot U^{-\beta\Delta\mu})^{\frac{1}{\eta\gamma\Delta\mu}}$.

Therefore, if there exists a non-negligible difference between the adversary's probabilities of success in Game 2 and Game 3, we can construct a simulator to solve the CDH problem.

Game 4. Game 4 is the same as Game 3, with only one difference. The challenger observes each instance of the cloud auditing protocol with the adversary. If in any instance the adversary is successful but there exists one adversary's aggregate message μ is not equal to $\mu = \sum_{i\in I} v_i m_i$, then the challenger will abort.

Analysis. Suppose the file that causes the abort in time period j has name $name$, is composed by $m_1, \cdots, m_n$, and that the set of authenticators issued by challenger are $\Phi = (j, U, \{\sigma_i\}_{1\leq i\leq n}, \Omega_j = (R_{\mathbf{wj}|_1}, \cdots, R_{\mathbf{wj}|_t}))$. Let $(j, Chal = \{(i, v_i)\}_{i\in I})$ be the query that makes the challenger abort, and the proof of the adversary be $P = (j, U, \sigma', \mu', \Omega_j)$. Let the expected response that would have been got from an honest prover be $P = (j, U, \sigma, \mu, \Omega_j)$. Game 3 has guaranteed $\sigma = \sigma'$; it can be only the values μ' and μ that are different. Set $\Delta\mu = \mu' - \mu$; again, $\Delta\mu \neq 0$. If the adversary can make the challenger in Game 4 abort, then we can construct a simulator to solve the discrete logarithm problem.

The simulator is given a tuple g, v; his goal is to find x such that $v = g^x$. The simulator acts like Game 3 challenger, with the following differences:

In Setup phase, the simulator selects $\beta, \gamma \in_R Z_q^*$ and set $u = g^\beta v^\gamma$.

The simulator interacts with the adversary as the real protocol. If the adversary successes in the game in responding with aggregate message μ' that is different from the expected aggregate message μ, the simulator can solve the discrete logarithm problem. Because of the change made in Game 3 we know that $\sigma' = \sigma$. So we can know the following equations hold from the verification equations using μ' and μ,

$$\hat{e}(R \cdot \prod_{m=1}^{t} R_{\mathbf{wj}|_m}^{h_{\mathbf{wj}|_m}}, H_1(R)^{\sum_{i\in I} v_i}) \cdot \hat{e}(U, u^{\mu} \cdot \prod_{i\in I} H_3(name||i||j,U)^{v_i}) = \hat{e}(g, \sigma) = \hat{e}(g, \sigma') =$$
$$\hat{e}(R \cdot \prod_{m=1}^{t} R_{\mathbf{wj}|_m}^{h_{\mathbf{wj}|_m}}, H_1(R)^{\sum_{i\in I} v_i}) \cdot \hat{e}(U, u^{\mu'} \cdot \prod_{i\in I} H_3(name||i||j,U)^{v_i}).$$

We can get that $u^\mu = u^{\mu'} \Rightarrow (g^\beta v^\gamma)^{\Delta\mu} = 1 \Rightarrow v = g^{-\frac{\beta}{\gamma}}$.

Therefore, if there exists a non-negligible difference between the adversary's probabilities of success in Game 3 and Game 4, we can construct a simulator to solve the discrete logarithm problem.

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. As a result, if the CDH problem in G_1 is hard and the signature scheme $SSig$ used for file tags is existentially unforgeable, the verifier will reject except when the prover responds with correctly computed values in $P = (j, U, \sigma, \mu, \Omega_j)$. We complete the proof of the first phase.

After we complete the proof of the first phase, we can know the prover must respond with correct $P = (j, U, \sigma, \mu, \Omega_j)$ to pass the verification. Now we will construct a knowledge extractor to extract the whole challenged file blocks $m_{s_1}, \cdots, m_{s_c}$. The procedure is similar to that in [1]. By selecting independent coefficients $v_1, \cdots, v_c$ to execute challenge phase in the protocol on the same blocks $m_{s_1}, \cdots, m_{s_c}$ for c times, the extractor can get c independent linear equations in the variables $m_{s_1}, \cdots, m_{s_c}$. The extractor can extract $m_{s_1}, \cdots, m_{s_c}$ by solving these equations. We complete the proof of the second phase.

This completes the proof of Theorem 2.

Acknowledgment

The authors would like to thank for anonymous reviewers for their helpful suggestions to improve this paper and Prof. Fangguo Zhang for the discussion about the security analysis of our protocol.

References

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable Data Possession at Untrusted Stores,” *Proc. 14th ACM Conf. Computer and Comm. Security*, pp. 598-609, 2007.

[2] G. Ateniese, R.D. Pietro, L. V. Mancini, and G. Tsudik, “Scalable and Efficient Provable Data Possession,” *Proc. 4th International Conference on Security and Privacy in Communication Networks*, 2008

[3] F. Sebe, J. Domingo-Ferrer, A. Martinez-balleste, Y. Deswarte, and J. Quisquater, “Efficient Remote Data Integrity checking in Critical Information Infrastructures,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 8, pp. 1-6, 2008.

[4] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, “MR-PPDP: Multiple-Replica Provable Data Possession,” *Proc. 28th IEEE International Conference on Distributed Computing Systems*, pp. 411-420, 2008.

[5] H. Shacham and B. Waters, “Compact Proofs of Retrievability,” *Advances in Cryptology-Asiacrypt’08*, pp. 90-107, 2008.

[6] C. Wang, K. Ren, W. Lou, and J. Li, “Toward Publicly Auditable Secure Cloud Data Storage Services,” *IEEE Network*, vol. 24, no. 4, pp. 19-24, July/Aug. 2010.

[7] Y. Zhu, H. Wang, Z. Hu, G. J. Ahn, H. Hu, and S. S. Yau, “Efficient Provable Data Possession for Hybrid Clouds,” *Proc. 17th ACM Conference on Computer and Communications Security*, pp. 756-758, 2010.

[8] K. Yang and X. Jia, “Data Storage Auditing Service in Cloud Computing: Challenges, Methods and opportunities,” *World Wide Web*, vol. 15, no. 4, pp. 409-428, 2012.

[9] K. Yang and X. Jia, “An efficient and secure dynamic auditing protocol for data storage in cloud computing,” *IEEE Trans. Parallel and Distributed Systems*, Vol. 24, No. 9, pp. 1717-1726, 2013.

[10] C. Wang, S. Chow, Q. Wang, K. Ren, and W. Lou, “Privacy-Preserving Public Auditing for Secure Cloud Storage,” *IEEE Trans. Computers*, Vol. 62, No. 2, pp. 362-375, 2013.

[11] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, “Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing,” *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 5, pp. 847-859, May 2011.

[12] Y. Zhu, H.G. Ahn, H. Hu, S.S. Yau, H.J. An, and C.J. Hu, “Dynamic Audit Services for Outsourced Storages in Clouds,” *IEEE Trans. on Services Computing*, vol. 6, no. 2, pp. 409-428, 2013.

[13] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, “Dynamic provable data possession,” *Proc. of the 16th ACM conference on Computer and communications security*, pp. 213-222, 2009.

[14] H. Wang, “Proxy Provable Data Possession in Public Clouds,” *IEEE Trans. Services Computing*, Vol. 6, no. 4, pp. 551-559, 2013.

[15] B. Wang, B. Li, and H. Li. “Public auditing for shared data with efficient user revocation in the cloud,” *INFOCOM 2013 Proceedings IEEE*, pp. 2904-2912, 2013.

[16] H. Wang, Q. Wu , B. Qin, and J. Domingo-Ferrer, “Identity-based remote data possession checking in public clouds,” *IET Information Security*, vol.8, no. 2, pp. 114-121, March 2014.

[17] T. Stewart, “Security Policy and Key Management: Centrally Manage Encryption Key,” http://www.slideshare.net/Tina-stewart/security-policy-and-enterprise-key-management-from-vormetric. August, 2012.

[18] Microsoft, http://technet.microsoft.com/en-us/library/cc961626.aspx, 2014.

[19] FBI, http://www.fbi.gov/news/news_blog/is-your-computer-infected-with-dnschanger-malware, 2012.

[20] FBI, http://www.fbi.gov/news/stories/2011/april/botnet_041411, 2011.

[21] M. Bellare and S. Miner, “A forward-secure digital signature scheme,” *Advances in Cryptology-CRYPTO’99*, pp.431-448, 1999.

[22] Y. Dodis, J. Katz, S. Xu, and M. Yung, “Key-insulated public key cryptosystems,” *Advances in Cryptology-Eurocrypt’02*, pp. 65-82, 2002.

[23] G. Itkis and L. Reyzin, “SiBIR: Signer-base intrusion-resilient signatures,” *Advances in Cryptology-CRYPTO’02*, pp. 499-514, 2002.

[24] R. Canetti, S. Halevi and J. Katz, “A forward-secure public-key encryption scheme,” *Advances in Cryptology-EUROCRYPT’03*, pp. 255-271, 2003.

[25] F. Hu, C.H. Wu and J.D. Irwin, “A new forward secure signature scheme using bilinear maps,” *Cryptology ePrint Archive*, Report 2003/188, 2003.

[26] B.G. Kang, J.H. Park and S.G. Hahn, “A new forward secure signature scheme,” *Cryptology ePrint Archive*, Report 2004/183, 2004.

[27] J. Yu, F. Kong, X. Cheng, R. Hao, and G. Li, “One Forward-secure Signature Scheme Using Bilinear Maps and Its Applications,” *Information Sciences*, Vol. 279, pp. 60-76, 2014.

[28] J. Yu, R. Hao, F. Kong, X. Cheng, J. Fan, and Y. Chen, “Forward-Secure Identity-Based Signature: Security Notions and Construction,” *Information Sciences*, Vol. 181, Iss. 3, pp. 648-660, 2011.

[29] C. Gentry and A. Silverberg, “Hierarchical ID-based cryptography,” *Advances in Cryptology-Asiacryp’02*, pp. 548-566, 2002.

[30] A. Juels, J. Burton, and S. Kaliski, “PORs: Proofs of Retrievability for Large Files,” *Proc. 14th ACM Conf. Computer and Comm. Security*, pp. 584-597, 2007.

[31] Y. Dodis, S.P. Vadhan, and D. Wichs, “Proofs of Retrievability via Hardness Amplification,” *Proc. Theory of Cryptography Conf. Theory of Cryptography*, pp. 109-127,

2009.

[32] M.A. Shah, M. Baker, J.C. Mogul, and R. Swaminathan, "Auditing to Keep Online Storage Services Honest," *Proc. 11th USENIX Workshop Hot Topics in Operating Systems*, pp. 1-6, 2007.

[33] Y. Zhu, H. Hu, G. Ahn, and M. Yu, "Cooperative Provable Data Possession for Integrity Verification in Multi-Cloud Storage," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2231-2244, Dec. 2012.

[34] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious ram," *Advances in Cryptology-Eurocrypt'13*, pp. 279-295, 2013.

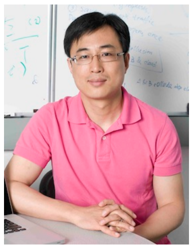
[35] E. Shi, E. Stefanov, and C. Papamanthou, "Practical dynamic proofs of retrievability," *Proc. 21st ACM Conf. Computer and Comm. Security*, pp. 325-336, 2013.

[36] N. Chandran, B. Kanukurthi, and R. Ostrovsky, "Locally updatable and locally decodable codes," *Proc. Theory of Cryptography 2014*, pp. 489-514, 2014.

[37] M. Etemad and A. Küpçü, "Transparent, distributed, and replicated dynamic provable data possession," *Proc. 11st Applied Cryptography and Network Security*. pp. 1-18, 2013.

[38] B. Chen and R. Curtmola, "Auditable Version Control Systems," *2014 Network and Distributed System Security Symposium*, 2014.

Jia Yu is a professor of the College of Information Engineering 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.



Kui Ren is an associate professor of Computer Science and Engineering and the director of the UbiSeC Lab at State University of New York at Buffalo. He received his PhD degree from Worcester Polytechnic Institute. Kui's current research interest spans Cloud & Outsourcing Security, Wireless & Wearable System Security, and Human-centered Computing. His research has been supported by NSF, DoE, AFRL, MSR, and Amazon. He is a recipient of NSF CAREER Award in 2011 and Sigma Xi/IIT Research Excellence Award in 2012. Kui has published 135 peer-review journal and conference papers and received several Best Paper Awards including IEEE ICNP 2011. He currently serves as an associate editor for IEEE Transactions on Mobile Computing, IEEE Transactions on Information Forensics and Security, IEEE Wireless Communications, IEEE Internet of Things Journal, IEEE Transactions on Smart Grid, Elsevier Pervasive and Mobile Computing, and Oxford The Computer Journal. Kui is a senior member of IEEE, a member of ACM, a Distinguished Lecturer of IEEE Vehicular Technology Society, and a past board member of Internet Privacy Task Force, State of Illinois.

Cong Wang is an Assistant Professor in the Computer Science Department at City University of Hong Kong. He received his B.E and M.E degrees from Wuhan University in 2004 and 2007, and PhD degree from Illinois Institute of Technology in 2012, all in Electrical and Computer Engineering. He has worked at Palo Alto Research Center in the summer of 2011. His research interests are in the areas of cloud computing and security, with current focus on secure data services in cloud computing, and secure computation outsourcing. He is a Member of the IEEE and a Member of the ACM.

Vijay Varadharajan is the Microsoft Chair Professor in Innovation in Computing Australia. He is also the Director of the Advanced Cyber Security Research Centre at Macquarie University, Sydney, Australia. Previously Vijay headed Secure Systems Research worldwide for Hewlett-Packard Labs based at European Headquarters at HP Labs Bristol, UK. Vijay has published more than 350 papers in international journals and conferences. Vijay has been/is on the Editorial Board of several journals including ACM Trans TISSEC, IEEE Trans TDSC, IEEE Trans TIFS, and IEEE Trans TCC. Vijay has supervised successfully over 30 PhD research students in the UK and Australia. He has had several visiting positions at different institutions in the world, including Isaac Newton Institute at Cambridge University, at Microsoft Research Labs UK/USA, INRIA Research Labs France, Edinburgh University, National University of Singapore, Chinese Academy of Sciences and Indian Inst. of Technology. Vijay is a Fellow of the British Computer Society (FBCS), a Fellow of the IEE/IET, a Fellow of the Institute of Mathematics, UK (FIMA), a Fellow of the Australian Institute of Engineers (FIEAust) and a Fellow of the Australian Computer Society (FACS).