Enabling Cloud Storage Auditing with Verifiable Outsourcing of Key Updates

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

***Abstract*—Key-exposure resistance has always been an important issue for in-depth cyber defence in many security applications. Recently, how to deal with the key exposure problem in the settings of cloud storage auditing has been proposed and studied. To address the challenge, existing solutions all require the client to update his secret keys in every time period, which may inevitably bring in new local burdens to the client, especially those with limited computation resources such as mobile phones.**

In this paper, we focus on how to make the key updates as transparent as possible for the client and propose a new paradigm called cloud storage auditing with verifiable outsourcing of key updates. In this paradigm, key updates can be safely outsourced to some authorized party, and thus the key-update burden on the client will be kept minimal. Specifically, we leverage the third party auditor (TPA) in many existing public auditing designs, let it play the role of authorized party in our case, and make it in charge of both the storage auditing and the secure key updates for key-exposure resistance. In our design, TPA only needs to hold an encrypted version of the client's secret key, while doing all these burdensome tasks on behalf of the client. The client only needs to download the encrypted secret key from the TPA when uploading new files to cloud. Besides, our design also equips the client with capability to further verify the validity of the encrypted secret keys provided by TPA. All these salient features are carefully designed to make the whole auditing procedure with key exposure resistance as transparent as possible for the client. We formalize the definition and the security model of this paradigm. The security proof and the performance simulation show that our detailed design instantiations are secure and efficient.

***Index Terms*—Cloud storage; outsourcing computing; cloud storage auditing; key update; verifiability**

I. Introduction

CLOUD computing, as a new technology paradigm with promising further, is becoming more and more popular nowadays. It can provide users with unlimited computing resource. Enterprises and people can outsource time-consuming computation workloads to cloud without spending the extra capital on deploying and maintaining hardware and software. In recent years, outsourcing computation has attracted much attention and been researched widely. It has been considered in many applications including scientific computations [1], linear algebraic computations [2], linear programming computations [3] and modular exponentiation computations [4], etc. Besides, cloud computing can also provide users with unlimited storage resource. Cloud storage is universally viewed as one of the most important services of cloud computing. Although cloud storage provides great benefit to users, it brings new security challenging problems. One important security problem is how to efficiently check the integrity of the data stored in cloud. In recent years, many auditing protocols for cloud storage have been proposed to deal with this problem. These protocols focus on different aspects of cloud storage auditing such as the high efficiency [5–17], the privacy protection of data [18], the privacy protection of identities [19], dynamic data operations[13, 15, 16, 20], the data sharing [21, 22], etc.

The key exposure problem, as another important problem in cloud storage auditing, has been considered [23] recently. The problem itself is non-trivial by nature. Once the client's secret key for storage auditing is exposed to cloud, the cloud is able to easily hide the data loss incidents for maintaining its reputation, even discard the client's data rarely accessed for saving the storage space. The authors in [23] constructed a cloud storage auditing protocol with key-exposure resilience by updating the user's secret keys periodically. In this way, the damage of key exposure in cloud storage auditing can be reduced. But it also brings in new local burdens for the client because the client has to execute the key update algorithm in each time period to make his secret key move forward. For some clients with limited computation resources, they might not like doing such extra computations by themselves in each time period. It would be obviously more attractive to make key updates as transparent as possible for the client, especially in frequent key update scenarios. In this paper, we consider achieving this goal by outsourcing key updates.

However, it needs to satisfy several new requirements to achieve this goal. Firstly, the real client's secret keys for cloud storage auditing should not be known by the authorized party who performs outsourcing computation for key updates. Otherwise, it will bring the new security threat. So the authorized party should only hold an encrypted version of the user's secret key for cloud storage auditing. Secondly, because the authorized party performing outsourcing computation only knows the encrypted secret keys, key updates should be completed under the encrypted state. In other words, this authorized party should be able to update secret keys for cloud storage auditing from the encrypted version he holds. Thirdly, it should be very efficient for the client to recover the real secret key from the encrypted version that is retrieved from the authorized party. Lastly, the client should be able to verify the validity of the encrypted secret key after the client retrieves it from the authorized party. The goal of this paper is to design a cloud storage auditing protocol that can satisfy

J. Yu is with the College of Information Engineering, Qingdao University, Qingdao 266071, P.R China and with the Department of Computer Science and Engineering, The State University of New York at Buffalo. E-mail:qduyujia@gmail.com.

K. Ren is with 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.

above requirements to achieve the outsourcing of key updates. The main contributions are as follows:

(1) We propose a new paradigm called cloud storage auditing with verifiable outsourcing of key updates. In this new paradigm, key-update operations are not performed by the client, but by an authorized party. The authorized party holds an encrypted secret key of the client for cloud storage auditing and updates it under the encrypted state in each time period. The client downloads the encrypted secret key from the authorized party and decrypts it only when he would like to upload new files to cloud. In addition, the client can verify the validity of the encrypted secret key.

(2) We design the first cloud storage auditing protocol with verifiable outsourcing of key updates. In our design, the third-party auditor (TPA) plays the role of the authorized party who is in charge of key updates. In addition, similar to traditional public auditing protocols [11–17], another important task of the TPA is to check the integrity of the client's files stored in cloud. The TPA does not know the real secret key of the client for cloud storage auditing, but only holds an encrypted version. In the detailed protocol, we use the blinding technique with homomorphic property to form the encryption algorithm to encrypt the secret keys held by the TPA. It makes our protocol secure and the decryption operation efficient. Meanwhile, the TPA can complete key updates under the encrypted state. The client can verify the validity of the encrypted secret key when he retrieves it from the TPA. Therefore, the designed protocol satisfies the above mentioned four requirements.

(3) We formalize the definition and the security model of the cloud storage auditing protocol with verifiable outsourcing of key updates. We also prove the security of our protocol in the formalized security model and justify its performance by concrete implementation.

A. Related Work

Outsourcing Computation. How to effectively outsource time-consuming computations has become a hot topic in the research of the theoretical computer science in the recent two decades. Outsourcing computation has been considered in many application domains. Chaum and Pedersen [24] firstly proposed the notion of wallet databases with observers, in which a hardware was used to help the client perform some expensive computations. The method for secure outsourcing of some scientific computations was proposed by Atallah *et al.* [1]. Chevallier-Mames *et al.* [25] designed the first effective algorithm for secure delegation of elliptic-curve pairings based on an untrusted server. The first outsourcing algorithm for modular exponentiations was proposed by Hohenberger and Lysyanskaya [26], which was based on the methods of precomputation and server-aided computation. Atallah and Li [27] proposed a secure outsourcing algorithm to complete sequence comparisons. Chen *et al.* [4] proposed new algorithms for secure outsourcing of modular exponentiations. Benjamin and Atallah [2] researched on how to securely outsource the computation for linear algebra. Atallah and Frikken [28] gave further improvement based on the weak secret hiding assumption. Wang *et al.* [3] presented an efficient method for secure outsourcing of linear programming computation. Chen *et al.* [29] proposed an outsourcing algorithm for attribute-based signatures computations. Zhang *et al.* [30] proposed an efficient method for outsourcing a class of homomorphic functions.

Cloud Storage Auditing. How to check the integrity of the data stored in cloud is a hot topic in cloud security. The notion of "provable data possession" (PDP) was firstly proposed by Ateniese *et al.* [5] to ensure data possession at untrusted servers. The notion of "proof of retrievability" (PoR) was proposed by Juels and Kaliski Jr. [6] to ensure both possession and retrievability of data at untrusted servers. Wang *et al.* [18] proposed a public privacy-preserving auditing protocol. They used the random masking technique to make the protocol achieve privacy-preserving property. Proxy provable data possession protocol was proposed in [17]. The auditing protocols supporting dynamic data operations were also proposed in [13, 20]. Yang and Jia [16] proposed an auditing protocol supporting both the dynamic property and the privacy preserving property. The privacy preserving of the user's identity for shared data auditing was considered in [19]. The problem of user revocation in shared data auditing was considered in [21]. Yuan *et al.* [22] proposed a public auditing protocol for data sharing with multiuser modification.

All above auditing protocols are all built on the assumption that the secret key of the client is absolutely secure and would not be exposed. In [23], the authors firstly considered the key exposure problem in cloud storage auditing and proposed a cloud storage auditing protocol with key-exposure resilience. In that protocol, the secret keys for cloud storage auditing are updated periodically. As a result, any dishonest behaviors, such as deleting or modifying the client's data previously stored in cloud, can all be detected, even if the cloud gets the client's current secret key for cloud storage auditing. However, the client needs to update his secret key in each time period. It will add obvious computation burden to the client, especially when key updates are very frequent.

B. Organization

The rest paper is organized as follows: In Section 2, we introduce the system model, definitions, and preliminaries of our work. Then, we give a concrete description of our protocol in Section 3. Security and performance are analyzed in Section 4. We conclude the paper in Section 5. The detailed security proof is shown in the appendix.

II. Model, Definitions and Preliminaries

A. Model

We show the system model for cloud storage auditing with verifiable outsourcing of key updates in Fig. 1. There are three parties in the model: the client, the cloud and the third-party auditor (TPA). The client is the owner of the files that are uploaded to cloud. The total size of these files is not fixed, that is, the client can upload the growing files to cloud in different time points. The cloud stores the client's files and provides download service for the client. The TPA plays two important roles: the first is to audit the data files stored in cloud

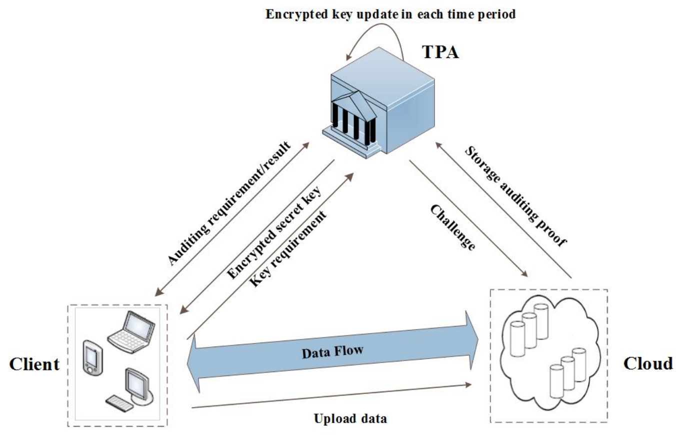


Fig. 1. System model of our cloud storage auditing

for the client; the second is to update the encrypted secret keys of the client in each time period. The TPA is thought to have powerful computational capability (its computational capability can be comparable with the cloud's). Similar to [23], the whole lifetime of the files stored in cloud is divided into $T+1$ time periods (from 0-th to T-th time periods). Each file is assumed to be divided into multiple blocks. In order to simplify the description, we do not furthermore divide each block into multiple sectors [7] in the description of our protocol. In the end of each time period, the TPA updates the encrypted client's secret key for cloud storage auditing according to the next time period. But the public key keeps unchanged in the whole time periods. The client sends the key requirement to the TPA only when he wants to upload new files to cloud. And then the TPA sends the encrypted secret key to the client. After that, the client decrypts it to get his real secret key, generates authenticators for files, and uploads these files along with authenticators to cloud. In addition, the TPA will audit whether the files in cloud are stored correctly by a challenge-response protocol between it and the cloud at regular time.

B. Definitions

(1) The definition of cloud storage auditing protocol with verifiable outsourcing of key updates

Definition 1: A cloud storage auditing protocol with secure outsourcing of key updates is composed by seven algorithms (*SysSetup*, *EkeyUpdate*, *VerESK*, *DecESK*, *AuthGen*, *ProofGen*, *ProofVerify*), shown below:

1) $SysSetup$: the system setup algorithm is run by the client. It takes as input a security parameter k and the total number of time periods T, and generates an encrypted initial client's secret key ESK_0, a decryption key DK and a public key PK. Finally, the client holds DK, and sends ESK_0 to the TPA.
2) $EkeyUpdate$: the encrypted key update algorithm is run by the TPA. It takes as input an encrypted client's secret key ESK_j, the current period j and the public key PK, and generates a new encrypted secret key ESK_{j+1} for period $j+1$.
3) $VerESK$: the encrypted key verifying algorithm is run by the client. It takes as input an encrypted client's secret key ESK_j, the current period j and the public key PK, if ESK_j is a well-formed encrypted client's secret key, returns 1; otherwise, returns 0.
4) $DecESK$: the secret key decryption algorithm is run by the client. It takes as input an encrypted client's secret key ESK_j, a decryption key DK, the current period j and the public key PK, returns the real client's secret key SK_j in this time period.
5) $AuthGen$: the authenticator generation algorithm is run by the client. It takes as input a file F, a client's secret key SK_j, the current period j and the public key PK, and generates the set of authenticators Φ for F in time period j.
6) $ProofGen$: the proof generation algorithm is run by the cloud. It takes as input a file F, a set of authenticators Φ, a challenge $Chal$, a time period j and the public key PK, and generates a proof P which proves the cloud stores F correctly.
7) $ProofVerify$: the proof verifying algorithm is run by the TPA. It takes as input a proof P, a challenge $Chal$, a time period j, and the public key PK, and returns "$True$" if P is valid; or "$False$", otherwise.

(2)Definition of Security

As same as other cloud storage auditing protocols [5–7, 9–13, 15–18, 20], the malicious cloud is viewed as the adversary in our security model. We use three games (Game 1, Game 2 and Game 3) to describe the adversaries with different compromising abilities who are against the security of the proposed protocol. Specifically, Game 1 describes an adversary, who fully compromises the TPA to get all encrypted secret keys ESK_j (periods $j=0,...,T$), tries to forge a valid authenticator in any time period. This game, in fact, shows the security should satisfy that the TPA cannot help the cloud to forge any authenticator in any time period even if it knows the encrypted secret keys. Game 2 describes an adversary, who compromises the client to get DK, tries to forge a valid authenticator in any time period. This game, in fact, shows the security should satisfy that an adversary cannot forge any authenticator in any time period even if it gets the decryption secret key DK by attacking the client. Game 3 provides the adversary more abilities, which describes an adversary, who compromises the client and the TPA to get both ESK_j and DK at one time period j, tries to forge a valid authenticator before time period j. This game, in fact, shows the security should satisfy that an adversary cannot forge any authenticator prior to one certain time period if it attacks the TPA and the client simultaneously to get their secret keys in this time period.

Game 1 is composed of four phases.

1) **Setup phase**. The challenger runs the system setup algorithm to generate the initial encrypted client's secret key ESK_0, the decryption key DK and the public key PK. The challenger updates the encrypted keys to get $ESK_1,...,ESK_T$ by running $EkeyUpdate$ algorithm. The challenger sends $ESK_0,...,ESK_T$ and PK to an

adversary $\mathcal{A}$, and holds DK himself. Set time period $j = 0$.

2) **Query phase**. $\mathcal{A}$ running in his phase can adaptively query the authenticators of the blocks specified by it in time period j as follows.
It adaptively selects and sends a series of blocks $m_1,\cdots,m_n$ to the challenger. The challenger runs $DecESK$ to decrypt ESK_j, computes the authenticators for $m_i (i = 1, \cdots, n)$ in time period j, and sends these authenticators to $\mathcal{A}$. $\mathcal{A}$ stores these blocks and authenticators. Set time period $j = j + 1$.
At the end of each time period, $\mathcal{A}$ can select to stay in this phase or go to the next phase.

3) **Challenge phase**. The challenger sends the adversary $\mathcal{A}$ a challenge $Chal$ and a time period j^*. The adversary is required to produce 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^*. If ProofVerify(PK, j^*, $Chal$, P)="$True$", then $\mathcal{A}$ wins in above game.

Game 2 is similar to the Game 1 except that the **Setup phase** is a little different. In the **Setup phase** of Game 2, the challenger needs to provides $\mathcal{A}$ with DK and PK instead of $ESK_0, ..., ESK_T$ and PK in Game 1. Specifically, **Game2** is also composed of four phases.

1) **Setup phase**. The challenger runs the system setup algorithm to generate the initial encrypted client's secret key ESK_0, the decryption key DK and the public key PK. The challenger sends DK and PK to an adversary $\mathcal{A}$, and holds ESK_0 himself. Set time period $j = 0$.

The **phases 2)-4)** are the same as those in Game 1.

Game 3 is different from Game 1 and Game 2, in which the adversary is allowed to get encrypted secret key and decryption key simultaneously at a certain time period. Specifically, Game 3 is composed of five phases.

1) **Setup phase**. The challenger runs the system setup algorithm to generate the initial encrypted client's secret key ESK_0, the decryption key DK and the public key PK. The challenger sends PK to an adversary $\mathcal{A}$, and holds DK and ESK_0 himself. Set time period $j = 0$.
2) **Query phase**. As same as Game 1.
3) **Break-in phase**. Set the break-in time period $b = j$. The challenger sends ESK_b and DK to the adversary. It means $\mathcal{A}$ can execute $DecESK$ algorithm to recover the secret key SK_b.
4) **Challenge phase**. The challenger sends the adversary $\mathcal{A}$ a challenge $Chal$ and a time period $j^* (j^* < b)$. The adversary is required to produce 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 $1 \leq c \leq n$.
5) **Forgery phase**. $\mathcal{A}$ outputs a proof of possession P for the blocks indicated by $Chal$ in time period j^*. If ProofVerify(PK, j^*, $Chal$, P)="$True$", then $\mathcal{A}$ wins in above game.

The above security model formalizes the adversaries with different reasonable abilities who try to cheat the challenger that he owns one file he in fact does not entirely know. We give Definition 2 to show that knowledge extractors can extract the challenged file blocks when adversaries output valid proofs in above games. Definition 3 describes the detectability for cloud storage auditing protocol, which shows that the cloud actually keeps the blocks that are not challenged with high probability. Definition 4 describes the verifiability for cloud storage auditing protocol, which shows that the client can verify whether the TPA provides the well-formed encrypted secret key.

Definition 2: We say a cloud storage auditing protocol with verifiable outsourcing of key updates is secure if the following condition holds: whenever an adversary $\mathcal{A}$ in above games that can cause the challenger to accept its proof with non-negligible probability, there exists efficient knowledge extractors that can extract the challenged file blocks except possibly with negligible probability.

Definition 3: A cloud storage auditing protocol with verifiable outsourcing of key updates is called a (ρ, δ) detectable protocol $(0 < \rho, \delta < 1)$ if, given a fraction ρ of corrupted blocks, the probability that the corrupted blocks are found is at least δ.

Definition 4: We say the cloud storage auditing protocol with outsourcing of key updates is verifiable if the valid encrypted secret keys provided by the TPA will always pass the client's verification, while the invalid ones will always be detected.

C. Preliminaries

Definition 5: Let G_1 and G_2 be two multiplicative cyclic group with the same prime order q. A bilinear pairing is a map $\hat{e} : G_1 \times G_1 \rightarrow G_2$ which satisfies:

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

Definition 6: Assume g is a generator of a multiplicative group with the prime order q. The CDH problem is that compute g^{ab} when given g^a and g^b $(a, b \in_R Z_q^*)$.

III. Our Proposed Protocol

A. High-level Technique Explanation

Our design is based on the structure of the protocol proposed in [23]. So we use the same binary tree structure as [23] to evolve keys. This tree structure can make the protocol achieve fast key updates and short key size. One important difference between the proposed protocol and the protocol in [23] is that the proposed protocol uses the binary tree to update the encrypted secret keys rather than the real secret keys. One problem we need to resolve is that the TPA should perform the outsourcing computations for key updates under the condition that the TPA does not know the real secret key of the client. Traditional encryption technique is not suitable

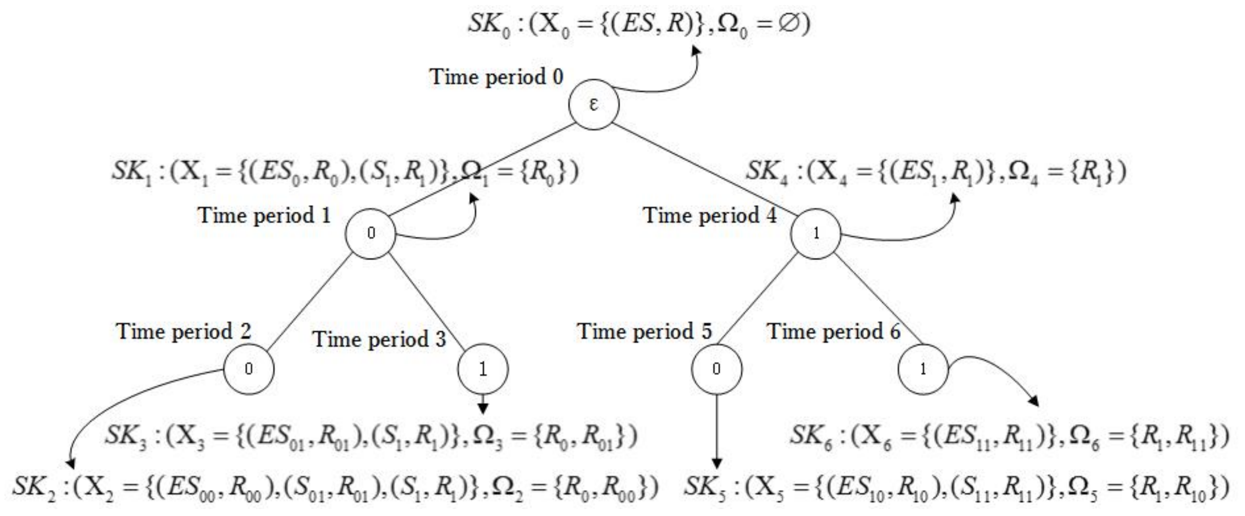

$SK_0 : (X_0 = \{(ES, R)\}, \Omega_0 = \varnothing)$
Time period 0
ε
$SK_1 : (X_1 = \{(ES_0, R_0), (S_1, R_1)\}, \Omega_1 = \{R_0\})$
$SK_4 : (X_4 = \{(ES_1, R_1)\}, \Omega_4 = \{R_1\})$
Time period 1
0
Time period 4
1
Time period 2
0
Time period 3
1
Time period 5
0
Time period 6
1
$SK_3 : (X_3 = \{(ES_{01}, R_{01}), (S_1, R_1)\}, \Omega_3 = \{R_0, R_{01}\})$
$SK_6 : (X_6 = \{(ES_{11}, R_{11})\}, \Omega_6 = \{R_1, R_{11}\})$
$SK_2 : (X_2 = \{(ES_{00}, R_{00}), (S_{01}, R_{01}), (S_1, R_1)\}, \Omega_2 = \{R_0, R_{00}\})$
$SK_5 : (X_5 = \{(ES_{10}, R_{10}), (S_{11}, R_{11})\}, \Omega_5 = \{R_1, R_{10}\})$

Fig. 2. An example show time periods and the corresponding secret keys

because it makes the key update difficult to be completed under the encrypted condition. Besides, it will be even more difficult to enable the client with the verification capability to ensure the validity of the encrypted secret keys. To address these challenges, we propose to explore the blinding technique with homomorphic property to efficiently "encrypt" the secret keys. It allows key updates to be smoothly performed under the blinded version, and further makes verifying the validity of the encrypted secret keys possible. Our security analysis later on shows that such blinding technique with homomorphic property can sufficiently prevent adversaries from forging any authenticator of valid messages. Therefore, it helps to ensure our design goal that the key updates are as transparent as possible for the client. In the designed $SysSetup$ algorithm, the TPA only holds an initial encrypted secret key and the client holds a decryption key which is used to decrypt the encrypted secret key. In the designed $KeyUpdate$ algorithm, homomorphic property makes the secret key able to be updated under encrypted state and makes verifying the encrypted secret key possible. The $VerESK$ algorithm can make the client check the validity of the encrypted secret keys immediately. In the end of this section, we will discuss the technique about how to make this check done by the cloud if the client is not in urgent need to know whether the encrypted secret keys are correct or not.

B. Notations and Structures

In Table 1, we show some notations used in the description of our protocol. The whole lifetime of files stored in cloud is divided into discrete time periods $0, ..., T$, and the same full binary tree with depth l as in [23] is used to appoint these time periods. We associate each period with each node of the tree by pre-order traversal technique, so total $2^l - 1$ periods (here $T = 2^l - 2$) can be associated with this binary tree. Begin with root node $w^0 = \varepsilon$. If w^j is an internal node, then $w^{j+1} = w'0$; if w^j is a leaf node, then $w^{j+1} = w'1$, where w' is the longest string such that $w'0$ is a prefix of w^j. Each node, corresponding to time period j, in the binary tree has one key pair (ES_{w^j}, R_{w^j}).

The TPA holds the client's encrypted secret key ESK_j in period j, which is composed by two parts X_j and Ω_j. The first part X_j is a set composed by the key pair (ES_{w^j}, R_{w^j}) and the key pairs of the right siblings of the nodes on the path from the root to w^j. That is, X_j comprises the secret key $(ES_{w'1}, R_{w'1})$ if and only if $w'0$ is a prefix of w^j. We use a stack to organize the first part X_j, which is initially set (ES, R) on top in time period 0. The second part Ω_j comprises the verification values from the root to node w^j except the root. So $\Omega_j = (R_{w^j|_1}, \cdots, R_{w^j|_t})$ if $w^j = w_1 \cdots w_t$. Fig. 2 shows an example about how to use a binary tree with depth $l = 2$ to associate time periods and the corresponding secret keys.

C. Description of the Protocol

Let G_1 and G_2 be two groups with some prime order q, where the CDH problem is difficult in G_1. Let g be a generator of G_1, and $\hat{e} : G_1 \times G_1 \rightarrow G_2$ be a bilinear pairing. Let $H_1 : G_1 \rightarrow G_1$, $H_2 : \{0,1\}^* \times G_1 \rightarrow Z_q^*$ and $H_3 : \{0,1\}^* \times G_1 \rightarrow G_1$ be three cryptographic hash functions. Following the same assumption as [23], the client has held a secret key for a signature $SSig$, which is used to ensure the integrity of not only the file identifier $name$ but also the time period j. Below we give the detailed description of our protocol.

1) **Algorithm SysSetup**: Input a security parameter k and the total time period T. Then
   a) The client selects $\rho, \tau \in_R Z_q^*$, and computes $R = g^\rho$, $G = g^\tau$ and $ES = H_1(R)^{\rho - \tau}$.
   b) The client sets $X_0 = \{(ES, R)\}$ and $\Omega_0 = \varnothing$(where $\varnothing$ is null set), and sends the initial encrypted secret keys $SK_0 = (X_0, \Omega_0)$ to the TPA. The client sets $DK = \tau$, and keeps it himself. The client randomly selects a generator u of group G_1.
   c) The public key is $PK = (R, G, u)$. Delete all intermediate data.

TABLE I
NOTATIONS

Notation	Meaning
T	The total periods number of the whole lifetime for the files stored in the cloud
$w_1...w_t$	The binary string of the node w^j associated with period j
$w^j\|_k (k \leq t)$	The k-prefix of w^j
$w^j0(w^j1)$	The left child node and the right child node of w^j
$w^j\|_{\bar{k}}$	The sibling node of $w^j\|_k$
ϵ	Empty binary string
PK	The public key which is unchanged in the whole lifetime
ES_{w^j}	The encrypted node secret key
R_{w^j}	The verification value which is used to verify the validity of authenticators
ESK_j	The client's encrypted secret key in period j
X_j	A set composed by the key pairs
Ω_j	A set composed by the verification values
(ES, R)	The key pair of the root node
F	A file which the client wants to store in cloud
$m_i(i = 1, ..., n)$	n blocks of file F
DK	The decryption key to recover the encrypted secret key for cloud storage auditing

2) **Algorithm EkeyUpdate**: Input an encrypted secret key ESK_j, the current time period j, and the public key PK.
Parse $ESK_j = (X_j, \Omega_j)$. X_j is organized as a stack which consists of (ES_{w^j}, R_{w^j}) and the key pairs of the right siblings of the nodes on the path from the root to w^j . The top element of the stack is (ES_{w^j}, R_{w^j}). Firstly, pop (ES_{w^j}, R_{w^j}) off the stack. Then do as follows:
   a) If w^j is an internal node ($w^{j+1} = w^j0$ in this case), select $\rho_{w^j0}, \rho_{w^j1} \in_R Z_q^*$. And then compute $R_{w^j0} = g^{\rho_{w^j0}}$, $R_{w^j1} = g^{\rho_{w^j1}}$, $h_{w^j0} = H_2(w^j0, R_{w^j0})$, $h_{w^j1} = H_2(w^j1, R_{w^j1})$, $ES_{w^j0} = ES_{w^j} \cdot H_1(R)^{\rho_{w^j0}h_{w^j0}}$ and $ES_{w^j1} = ES_{w^j} \cdot H_1(R)^{\rho_{w^j1}h_{w^j1}}$. Push (ES_{w^j1}, R_{w^j1}) and (ES_{w^j0}, R_{w^j0}) onto the stack orderly. Let X_{j+1} denote the current stack and define $\Omega_{j+1} = \Omega_j \bigcup \{R_{w^j0}\}$.
   b) If w^j is a leaf, define X_{j+1} with the current stack.
      i) If $w_t = 0$ (the node w^{j+1} is the right sibling node of w^j in this case), then set $\Omega_{j+1} = \Omega_j \bigcup \{R_{w^{j+1}}\} - \{R_{w^j}\}$ ($R_{w^{j+1}}$ can be read from the new top $(ES_{w^{j+1}}, R_{w^{j+1}})$ of the stack).
      ii) If $w_t = 1$ ($w^{j+1} = w''1$ in this case, where w'' is the longest string such that $w''0$ is a prefix of w^j), then set $\Omega_{j+1} = \Omega_j \bigcup \{R_{w^{j+1}}\} - \{R_{w''0}, R_{w''01}, \cdots, R_{w_t}\}$ ($R_{w^{j+1}}$ can be read from the new top $(ES_{w^{j+1}}, R_{w^{j+1}})$ of the stack).
   c) Finally, erase key pair (ES_{w^j}, R_{w^j}), and return $ESK_{j+1} = (X_{j+1}, \Omega_{j+1})$.
3) **Algorithm VerESK**: Input a client's encrypted secret key $ESK_j = (X_j, \Omega_j)$, the current period j and the public key PK. Verify whether the following equation holds:
$$\hat{e}(g, ES_{w^j}) = \hat{e}(R/G \cdot \prod_{m=1}^{t} R_{w^j1}{}^{h_{w^j1}}, H_1(R)),$$
where $h_{w^j} = H_2(w^j, R_{w^j})$.
If above equation holds, return 1; otherwise, return 0.
4) **Algorithm DecESK**: Input an encrypted client's secret key ESK_j, a decryption key DK, the current period j, and the public key PK. The client decrypts the secret key as follows.
$$S_{w^j}=ES_{w^j} \cdot H_1(R)^{\tau}.$$
The real secret key is $SK_j = (X'_j, \Omega_j)$, where X'_j is the same stack as X_j except that the top element in X'_j is (S_{w^j}, R_{w^j}) instead of (ES_{w^j}, R_{w^j}) in X_j.
5) **Algorithm AuthGen**: Input a file $F = \{m_1, \cdots, m_n\}$, a client's secret key SK_j, the current period j and the public key PK.
   a) The client parses $SK_j = (X'_j, \Omega_j)$ and reads the top element (S_{w^j}, R_{w^j}) from the stack X'_j. The client selects $r \in_R Z_q^*$, and computes $U = g^r$ and $\sigma_i = H_3(name\|i\|j, U)^r \cdot S_{w^j} \cdot u^{rm_i}$ $(i = 1, \cdots, n)$, where the name $name$ is chosen randomly from Z_q^* as the identifier of file F. In order to ensure the integrity of $name$ and j, the client also generates a file tag for F and j using the signature $SSig$.
   b) Denote the set of authenticators in time period j with $\Phi = (j, U, \{\sigma_i\}_{1 \leq i \leq n}, \Omega_j)$.
   c) Finally, the client sends the file F and the set of authenticators along with the file tag to cloud.
6) **Algorithm ProofGen**: Input a file F, a set of authenticators $\Phi = (j, U, \{\sigma_i\}_{1 \leq i \leq n}, \Omega_j)$, 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 $v_i \in Z_q$) and the public key PK.
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 $\mu = \sum_{i \in I} v_i m_i$. It then sends $P = (j, U, \sigma, \mu, \Omega_j)$ along with the file tag as the response proof of storage correctness to the TPA.

7) **Algorithm ProofVerify**: Input a proof P, a challenge $Chal$, a time period j and the public key PK.

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

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

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

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

D. How to remove the encrypted secret key verification of the client

If the client is not in urgent need to know whether the encrypted secret keys downloaded from the TPA are correct, we can remove his verifying operations and make the cloud perform the verification operations later. In this case, we can delete the $VerEKey$ algorithm from our protocol. When the client updates blocks and authenticators to cloud, the cloud needs to verify whether the following equation holds for any block m_i.

$$\hat{e}(R/G \cdot \prod_{m=1}^{t} R_{w^j|_m}{}^{h_{w^j|_m}}, H_1(R)) \cdot \hat{e}(U, u^{m_i} \cdot H_3(name||i||j, U)) = \hat{e}(g, \sigma_i).$$

If it holds, then the encrypted secret key must be correct. In this way, the client does not need to verify the encrypted secret keys right after he downloads it from the TPA. As a result, the computation burden of the client can be further saved.

IV. Security and Performance

A. Security Analysis

Theorem 1 (Correctness): For each random challenge $Chal$ and one valid proof $P = (j, U, \sigma, \mu, \Omega_j)$, the ProofVerify algorithm always returns "$True$".

Proof. It is because the following equations hold:

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

Theorem 2 (Security): If the CDH problem in G_1 is hard, then the proposed cloud storage auditing protocol with verifiable outsourcing of key updates is secure.

Proof. According to Definition 2, this theorem can be deduced from the following three lemmas.

Lemma 1: If the CDH problem in G_1 is hard, whenever an adversary $\mathcal{A}$ in Game 1 that can cause the challenger to accept its proof with non-negligible probability, there exists efficient knowledge extractors that can extract the challenged file blocks except possibly with negligible probability.

Proof. Please see Appendix A.

Lemma 2: If the CDH problem in G_1 is hard, whenever an adversary $\mathcal{A}$ in Game 2 that can cause the challenger to accept its proof with non-negligible probability, there exists efficient knowledge extractors that can extract the challenged file blocks except possibly with negligible probability.

Proof. Please see Appendix B.

Lemma 3: If the CDH problem in G_1 is hard, whenever an adversary $\mathcal{A}$ in Game 3 that can cause the challenger to accept its proof with non-negligible probability, there exists efficient knowledge extractors that can extract the challenged file blocks except possibly with negligible probability.

Proof. Please see Appendix C.

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.

Proof. Assume that 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 TPA matches the bad blocks. Let X be a discrete random variable that is defined to be the number of blocks chosen by the challenger that matches the block-tag pairs modified by the adversary. The probability that at least one of the blocks picked by the challenger matches one of the blocks modified by the adversary is denoted as P_X. We have:

$$\begin{aligned}
P_X &= P\{X \geq 1\} \\
&= 1 - P\{X = 0\} \\
&= 1 - \frac{n-t}{n} \frac{n-1-t}{n-1} \cdot \ldots \cdot \frac{n-c+1-t}{n-c+1}
\end{aligned}$$

Thus, we can get $P_X \leq 1 - (\frac{n-t}{n})^c$.

Theorem 4 (Verifiability): The proposed cloud storage auditing protocol with outsourcing of key updates is verifiable.

Proof. If the encrypted secret key downloaded from the TPA is valid, then $ES_{w^j} = H_1(R)^{\rho-\tau} \prod_{m=1}^{t} H_1(R)^{\rho_{w^j 1} h_{w^j 1}}$. So

$$\begin{aligned}
&\hat{e}(g, ES_{w^j}) \\
&= \hat{e}(g, H_1(R)^{\rho-\tau} \prod_{m=1}^{t} H_1(R)^{\rho_{w^j 1} h_{w^j 1}}) \\
&= \hat{e}(g^{\rho-\tau} \cdot \prod_{m=1}^{t} g^{\rho_{w^j 1} h_{w^j 1}}, H_1(R)) \\
&= \hat{e}(R/G \cdot \prod_{m=1}^{t} R_{w^j 1}{}^{h_{w^j 1}}, H_1(R)).
\end{aligned}$$

It means the valid encrypted secret key can pass the $VerESK$ algorithm.

If the encrypted secret key downloaded from the TPA is invalid, then $ES_{w^j} \neq H_1(R)^{\rho-\tau} \prod_{m=1}^{t} H_1(R)^{\rho_{w^j 1} h_{w^j 1}}$. We can know $\hat{e}(g, ES_{w^j}) \neq \hat{e}(R/G \cdot \prod_{m=1}^{t} R_{w^j 1}{}^{h_{w^j 1}}, H_1(R))$. It means the client can detect it is invalid in the $VerESK$ algorithm.

Therefore, this theorem holds.

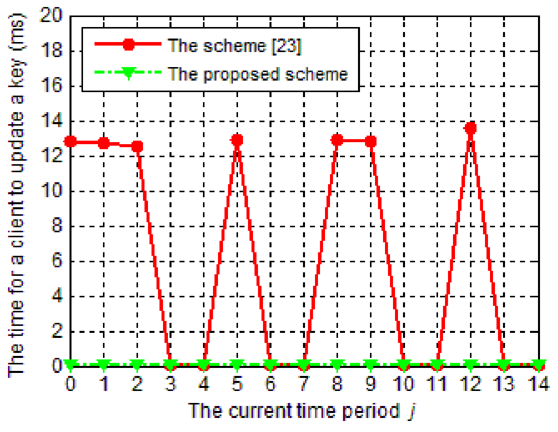


Fig. 3. The key update time on client side in the proposed scheme and the scheme [23]

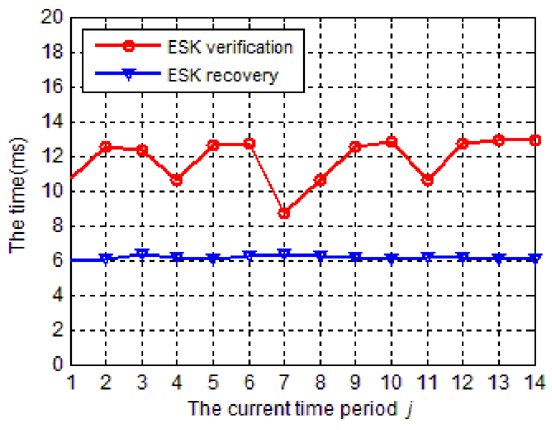


Fig. 4. The time to verify and recover an encrypted secret key (ESK) in different time periods

B. Performance Analysis

We evaluate the performance of the proposed scheme through several experiments that are implemented with the help of the Pairing-Based Cryptography (PBC) library [31]. We carry out these experiments on a Linux server with Intel processor running at 2.70 GHz and 4 GB memory. The elliptic curve picked to realize bilinear mapping is a supersingular curve with fast pairing operation, and the order of the curve group has 160 bits. In this case, the size of an element in Z_q^* is 20 bytes, and the size of an element in G_1 is 128 bytes. The data file in our experiments is 20 M comprising 1,000,000 blocks. For simplification, we set the total time period $T = 14$ which means we can use a full binary tree with depth 2 to associate with these time periods. All experimental results are taken as the mean values from multiple executions.

In the proposed scheme, the key update workload is outsourced to the TPA. In contrast, the client has to update the secret key by itself in each time period in scheme [23]. We compare the key update time on client side between the both schemes in Fig. 3. In scheme [23], the key update time on the client is related to the depth of the node corresponding to the current time period in binary tree. When the depth of node corresponding to the current time period is 0 or 1 (the node is internal node), the update time is about 12.6ms; when it is 2 (the node is leaf node), the update time is almost zero. In our scheme, the key update time on client side is zero in all time periods.

When the client wants to upload new files to the cloud, it needs to verify the validity of the encrypted secret key from the TPA and recover the real secret key. We show the time for these two processes happened in different time periods in Fig. 4. In practice, these processes do not happen in most of time periods. They only happen in the time periods when the client needs to upload new files to the cloud. Furthermore, the work for verifying the correctness of the encrypted secret key can fully be done by the cloud if we use the method in the end of Section 3.

We demonstrate the time of the challenge generation process, the proof generation process, and the proof verification

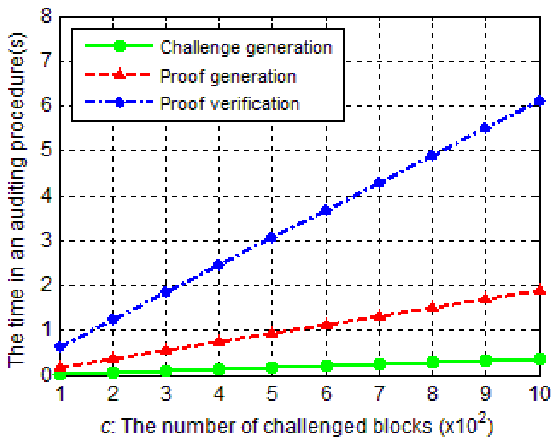
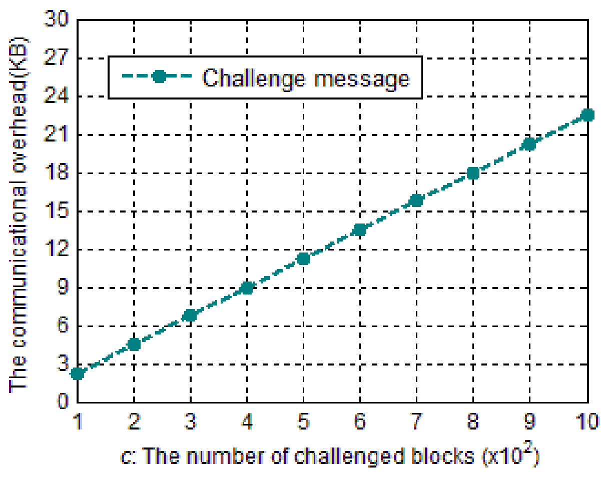


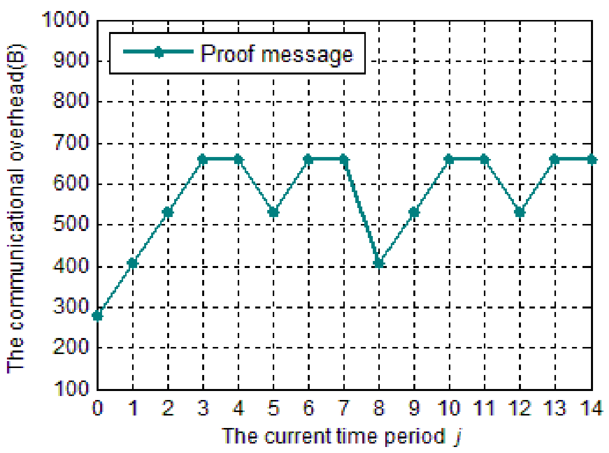
Fig. 5. The time of auditing procedures with different number of checked blocks

process with different number of checked data blocks in Fig. 5. In our evaluation, the number of checked block varies from 100 to 1,000. All the time of these processes increase linearly with the number of checked blocks. The challenge generation process spends the least time, which is 0.35s at most; the proof generation process spends more time, varying from 0.19s to 1.89s; the proof verification process spends the most time varying from 0.62s to 6.12s.

In our scheme, the communicational messages comprise the challenge message and the proof message. From Fig. 6 (a), we can see that the challenge message is linear with the number of checked blocks. The size of challenge message is 2.25 KB when the checked blocks are 100, and increases to 22.5 KB when the checked blocks are 1,000. As analyzed in [5], when the number of checked blocks is 460, the TPA can detect the data abnormity in the cloud with a probability at least 99%. In this case, the challenge message would be 10.35 KB. From Fig. 6 (b), we can see that the size of proof message varies with the depths of nodes corresponding to time periods. In period 0, the proof message is the shortest, which is 276.5 bytes, since the depth of the corresponding node is 0. And the longest proof messages appear at the leaves of the tree, which



(a)



(b)

Fig. 6. Communicational Cost. (a)The size of the challenge message with different number of checked blocks. (b)The size of the proof message in different time periods

is 0.66 KB.

V. Conclusion

In this paper, we study on how to outsource key updates for cloud storage auditing with key-exposure resilience. We propose the first cloud storage auditing protocol with verifiable outsourcing of key updates. In this protocol, key updates are outsourced to the TPA and are transparent for the client. In addition, the TPA only sees the encrypted version of the client's secret key, while the client can further verify the validity of the encrypted secret keys when downloading them from the TPA. We give the formal security proof and the performance simulation of the proposed scheme.

Appendix A
Proof of the Lemma 1

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

Game a. *Game a* is the same as Game 1, with only one difference. The challenger holds a list that contains signed tags ever issued as part of authenticator queries. If the adversary issues one tag, the challenger will abort when this tag is a valid signature produced by $SSig$ algorithm but not signed by the challenger.

Analysis. If the adversary can make the challenger abort in *Game a* with non-negligible probability, it is obvious that the adversary can be transformed to a forger against the $SSig$ signature algorithm. So it means that $name$, j and any verification value in Ω_j used in the interactions with the adversary are all generated by the challenger.

Game b. *Game b* is the same as *Game a*, with only one difference. The challenger holds a list to keep his responses to authenticator queries made by the adversary. If the adversary is successful in the game but U in the proof P is different from the actual U in the set of authenticators $\Phi = (j, U, \{\sigma_i\}_{1\leq i\leq n}, \Omega_j)$ that the challenger has kept, then the challenger will abort.

Analysis. If the adversary ia able to make the challenger in *Game b* abort, we will construct a simulator $\mathcal{S}$ that is used to solve the CDH problem in G_1. $\mathcal{S}$ behaves like the *Game a* challenger, only with the following differences:

Setup phase

Firstly, $\mathcal{S}$ 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 $\mathcal{S}$ is to compute $g^{a'b'}$, where $b' = \rho$ and $a', b' \in Z_q^*$ are unknown to $\mathcal{S}$ and $\mathcal{A}$. $\mathcal{S}$ randomly selects $\alpha \in Z_q^*$, and sets $ES = H_1(R)^{\alpha}$ and $G = R \cdot g^{-\alpha}$. It means $DK = \tau = b' - \alpha$, but $\mathcal{S}$ and $\mathcal{A}$ do not know it. $\mathcal{S}$ can know $ESK_0 = (X_0, \Omega_0)$, where $X_0 = \{(ES, R)\}$ and $\Omega_0 = \varnothing$. He selects $\beta \in_R Z_q^*$, and computes $u = g^{\beta}$. $\mathcal{S}$ executes $EkeyUpdate$ algorithm to generate $ESK_0, ..., ESK_T$. After this procedure, $\mathcal{S}$ has defined $\rho_{w^j|_m}$, and computed $h_{w^j|_m}(m = 1, ..., t)$. Then he provides $ESK_0, ..., ESK_T$ and $PK = (R, G, u)$ to $\mathcal{A}$.

Query phase

H_3 **queries.** $\mathcal{S}$ holds a list of queries. When $\mathcal{A}$ queries H_3 oracle at a point $< name||i||j, U >$, $\mathcal{S}$ does as follows.

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

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

(2)Otherwise, $\mathcal{S}$ selects $\lambda \in_R Z_q^*$, 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. $\mathcal{S}$ holds a list of queries. When $\mathcal{A}$ makes the authenticator query on block m_i with name $name$ in time period j, $\mathcal{S}$ does as follows.

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

(2) Secondly, he computes $h_{w^j|_m} = H_2(w^j|_m, R_{w^j|_m})$ and sets $\Omega_j = (R_{w^j|_1}, ..., R_{w^j|_t})$. He computes

$\psi = R^{\lambda_i\gamma+\beta\gamma m_i} \cdot {I_1}^{\sum_{m=1}^{t} h_{wj|_m} \rho_{wj|_m}}$.
It is because

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

.

(3) Finally, $\mathcal{S}$ responds $< j, U, \psi, \Omega_j) >$ to $\mathcal{A}$.

Challenge Phase

$\mathcal{S}$ randomly selects a challenge $Chal = \{(i, v_i)\}_{i\in I}$, where $I = \{s_1, s_2, \cdots, s_c\}$, and a time period j^*. And then $\mathcal{S}$ provides $\mathcal{A}$ with $Chal$ and j^*. $\mathcal{S}$ requests $\mathcal{A}$ to prove that $\mathcal{A}$ possesses the blocks $\{m_{s_1}, \cdots, m_{s_c}\}$ of file $F = \{m_1, \cdots, m_n\}$.

Forgery phase $\mathcal{A}$ outputs a proof $P = (j, U, \sigma, \mu, \Omega_j)$ as the response proof. If ProofVerify$(j^*, PK, Chal, P) =$ "$True$", then

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

If U in the $\mathcal{A}$'s proof $P = (j, U, \sigma, \mu, \Omega_j)$ is different from the real U in the set of authenticators $\Phi = (j, U, \{\sigma_i\}_{1\le i\le n}, \Omega_j)$ that $\mathcal{S}$ has kept, $\mathcal{S}$ abstracts $< name||i||j^*, U, h, \lambda_i, * >$ from H_3 table to get $H_3(name||i||j^*, U) = g^{\lambda_i}$ with high probability. It is obvious that the probability $\sum_{i\in I} v_i = 0$ is negligible. Using $h_{w^{j^*}|_m}$, $\rho_{w^{j^*}|_m}$ generated during the key update of Setup phase, $\mathcal{S}$ can solve the CDH problem as follows.

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

It means that U in prover's proof $P = (j, U, \sigma, \mu, \Omega_j)$ should be correct. So the difference between $\mathcal{A}$'s probability of success in *Game a* and *Game b* is negligible.

Game c. *Game c* is the same as *Game b*, with only one difference. The challenger holds a list that contains its responses to $\mathcal{A}$'s authenticator queries. The challenger observes each instance of the game with the adversary. If $\mathcal{A}$ succeeds in any instance but $\mathcal{A}$'s aggregate authenticator σ is not equal to the real aggregate authenticator $\sigma = \prod_{i\in I} \sigma_i^{v_i}$, then the challenger will abort.

Analysis. Let the file $F = \{m_1, \cdots, m_n\}$ with name $name$ causes the abort in time period j, and the corresponding set of authenticators provided by challenger are $\Phi = (j, U, \{\sigma_i\}_{1\le i\le n}, \Omega_j = (R_{\mathbf{wj}|_1}, \cdots, R_{\mathbf{wj}|_t}))$. Let $(j, Chal = \{(i, v_i)\}_{i\in I})$ be the query that causes the abort, and $P = (j, U, \sigma', \mu', \Omega_j)$ be the proof provided by the adversary. Let the expected response from an honest prover be $P = (j, U, \sigma, \mu, \Omega_j)$. The correctness of the proof means the following equation holds.

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

It is obvious that $\sigma \neq \sigma'$ and σ' can pass the verification equation because the challenger aborts. So

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

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

We now construct a simulator $\mathcal{S}$ that is used to solve the CDH problem in G_1 if the adversary $\mathcal{A}$ can make $\mathcal{S}$ abort with non-negligible probability. $\mathcal{S}$ acts like the *Game b* challenger, with the following differences:

Firstly, $\mathcal{S}$ is given a tuple $(g, g^{a'} = H_1(R), v = R)$; the final goal is to compute $v^{a'}$. In Setup phase, $\mathcal{S}$ randomly selects $\alpha \in Z_q^*$, and sets $ES = H_1(R)^{\alpha}$ and $G = R \cdot g^{-\alpha}$. It means $DK = \tau = b' - \alpha$, but $\mathcal{S}$ and $\mathcal{A}$ do not know it. $\mathcal{S}$ can know $ESK_0 = (X_0, \Omega_0)$, where $X_0 = \{(ES, R)\}$ and $\Omega_0 = \varnothing$. $\mathcal{S}$ sets $u = g^{\beta} v^{\gamma}$ for some $\beta, \gamma \in_R Z_q^*$. $\mathcal{S}$ executes $EkeyUpdate$ algorithm to generate $ESK_0, ..., ESK_T$, and provides $ESK_0, ..., ESK_T$ and $PK = (R, G, u)$ to $\mathcal{A}$.

$\mathcal{S}$ controls a random oracle H_3, and stores a list of queries. When $\mathcal{A}$ makes a H_3 oracle query at $< name||i||j, U >$, $\mathcal{S}$ will check if $< name||i||j, U >$ is in a tuple $< name||i||j, U, h, \lambda, \gamma >$ of H_3 table. If it is in, $\mathcal{S}$ provides h to $\mathcal{A}$; Otherwise, $\mathcal{S}$ selects $\lambda \in_R Z_q^*$, and computes $h = g^{\lambda}$. It adds tuple $< name||i||j, U, h, \lambda, * >$ to H_3 table and provides h to $\mathcal{A}$.

$\mathcal{S}$ holds a list of authenticator queries. If $\mathcal{A}$ queries the authenticator of one block m_i with name *name* in time period j, $\mathcal{S}$ selects $\lambda_i, \eta \in_R Z_q^*$, computes $U = (g^{a'})^{\eta}$, and sets $h = g^{\lambda_i} \cdot g^{-\frac{\alpha}{\eta}} \cdot (g^{b'})^{\frac{1}{\eta}} / (g^{\beta} v^{\gamma})^{m_i}$. According to our previous analysis, $H_3(name||i||j, U)$ has not been defined except possibly with negligible probability. $\mathcal{S}$ can compute authenticator σ_i and respond it to $\mathcal{A}$ because $\sigma_i = H_3(name||i||j, U)^{a'\eta} \cdot S_{\mathbf{wj}} \cdot u^{a'\eta m_i} = (g^{\lambda_i} \cdot g^{-\frac{\alpha}{\eta}} \cdot (g^{b'})^{\frac{1}{\eta}} / (g^{\beta} v^{\gamma})^{m_i})^{a'\eta} \cdot S_{\mathbf{wj}} \cdot ((g^{\beta} v^{\gamma})^{m_i})^{a'\eta} = (g^{a'})^{\lambda_i\eta} \cdot S_{\mathbf{wj}} \cdot (g^{a'})^{-\alpha} \cdot g^{a'b'} = (g^{a'})^{\lambda_i\eta} \cdot S_{\mathbf{wj}} \cdot H_1(R)^{-\alpha+b'} = (g^{a'})^{\lambda_i\eta} \cdot S_{\mathbf{wj}} \cdot H_1(R)^{-\tau} = (g^{a'})^{\lambda_i\eta} \cdot ES_{\mathbf{wj}}$. Here, $ES_{\mathbf{wj}}$ as a part of ESK_j has been known by $\mathcal{S}$ in Setup phase. Finally, $\mathcal{S}$ adds $< name||i||j, U, h, \lambda_i, \eta >$ to H_3 table.

If $\mathcal{A}$ responds with a proof $P = (j, U, \sigma', \mu', \Omega_j)$ in which σ' is different from the expected σ, and finally successes in the above game, $\mathcal{S}$ can extract an tuple $< name||i||j, U, h, \lambda_i, \eta >$ $(i \in I)$ from H_3 table. And then $\mathcal{S}$ 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 $\hat{e}(\sigma' \cdot \sigma^{-1} \cdot U^{-\beta\Delta\mu}, g) = \hat{e}((g^{a'})^{\eta}, v)^{\gamma\Delta\mu}$. $\mathcal{S}$ can solve the CDH problem $v^{a'} = (\sigma' \cdot \sigma^{-1} \cdot U^{-\beta\Delta\mu})^{\frac{1}{\eta\gamma\Delta\mu}}$.

Therefore, we can construct a simulator $\mathcal{S}$ to solve the CDH problem if there is a non-negligible difference between the adversary's probabilities of success in *Game b* and *Game c*.

Game d. *Game d* is the same as *Game c*, with only one difference. The challenger observes each instance of the game with the adversary. If the adversary succeeds in any instance but there is one adversary's aggregate message μ is not equal to the real aggregate message $\mu = \sum_{i\in I} v_i m_i$, then the challenger will abort.

Analysis. There exists a non-negligible difference between the adversary's probabilities of success in *Game c* and *Game d* as long as the CDH problem in G_1 is difficult. This analysis is similar to the analysis of Game 4 in [23]. Here, we omit it.

As a result, there is only negligible difference probability between these games.

And then, we can construct a knowledge extractor to extract all challenged file blocks $m_{s_1}, \cdots, m_{s_c}$. By using independent coefficients $v_1, \cdots, v_c$ to execute Challenge phase on the same blocks $m_{s_1}, \cdots, m_{s_c}$ for c times, the extractor obtains c independent linear equations in the variables $m_{s_1}, \cdots, m_{s_c}$. It is easy for the extractor to extract $m_{s_1}, \cdots, m_{s_c}$ by solving these equations.

APPENDIX B
PROOF OF THE LEMMA 2

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

Game a'. *Game a'* is the same as Game 2, with only one difference. The challenger holds a list that contains signed tags ever issued as part of authenticator queries. If the adversary issues one tag, the challenger will abort when this tag is a valid signature produced by $SSig$ algorithm but not signed by the challenger.

According to the analysis in APPENDIX A, the challenger cannot abort in *Game a'* with non-negligible probability.

Game b'. *Game b'* is the same as *Game a'*, with only one difference. The challenger holds a list to keep his responses to authenticator queries made by the adversary. If the adversary is successful in the game but U in the proof P is different from the actual U in the set of authenticators $\Phi = (j, U, \{\sigma_i\}_{1\le i\le n}, \Omega_j)$ that the challenger has kept, then the challenger will abort.

Analysis. If an adversary $\mathcal{S}$ is able to make the challenger in *Game b* abort, we will construct a simulator $\mathcal{S}$ that is used to solve the CDH problem in G_1. $\mathcal{S}$ behaves like the *Game a'* challenger, only with the following differences:

Setup phase

Firstly, $\mathcal{S}$ 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 $\mathcal{S}$ is to compute $g^{a'b'}$, where where $b' = \rho$ and $a', b' \in Z_q^*$ are unknown to $\mathcal{S}$ and $\mathcal{A}$. $\mathcal{S}$ selects $\tau, \beta \in_R Z_q^*$, and computes $G = g^\tau$ and $u = g^\beta$. Then $\mathcal{S}$ provides $PK = (R, G, u)$ and $DK = \tau$ to $\mathcal{A}$.

Query phase

In order to answer the queries from $\mathcal{A}$, $\mathcal{S}$ needs to make some preparation for key updates. We use $\mathbf{w^j} = w_1 \cdots w_t$ to denote the node associated with the current time period $j (j < b)$. He does as follows.

(1) if $\mathbf{w^j}$ is the leaf, $\mathcal{S}$ does nothing.

(2) Otherwise, $\mathcal{S}$ selects $\rho_{\mathbf{w^j}0}, \rho_{\mathbf{w^j}1}, h_{\mathbf{w^j}0}, h_{\mathbf{w^j}1} \in_R Z_q^*$, and computes $R_{\mathbf{w^j}0} = g^{h_{\mathbf{w^j}0}}$, $R_{\mathbf{w^j}1} = g^{h_{\mathbf{w^j}1}}$, $h_{\mathbf{w^j}0} = H_2(\mathbf{w^j}0, R_{\mathbf{w^j}0})$ and $h_{\mathbf{w^j}1} = H_2(\mathbf{w^j}1, R_{\mathbf{w^j}1})$.

H_3 **queries.** $\mathcal{S}$ holds a list of queries. When $\mathcal{A}$ queries H_3 oracle at a point $< name||i||j, U >$, $\mathcal{S}$ does as follows.

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

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

(2)Otherwise, $\mathcal{S}$ selects $\lambda \in_R Z_q^*$, 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. $\mathcal{S}$ holds a list of queries. When $\mathcal{A}$ makes the authenticator query on block m_i with name $name$ in time period j, $\mathcal{S}$ does as follows.

(1) Firstly, $\mathcal{S}$ selects $\gamma, \lambda_i \in_R Z_q^*$, and computes $U = R^\gamma$ and $h = g^{\lambda_i} \cdot I_1{}^{-1/\gamma}$. According to the previous analysis, the probability that the same $name$ and U have been chosen by $\mathcal{S}$ is negligible. $\mathcal{S}$ adds $< name||i||j, U, h, \lambda_i, \gamma >$ to H_3 table.

(2) Secondly, he sets $\Omega_j = (R_{w^j|_1}, ..., R_{w^j|_t})$ and computes $\psi = R^{\lambda_i\gamma + \beta\gamma m_i} \cdot I_1{}^{\sum_{m=1}^{t} h_{w^j|_m}\rho_{w^j|_m}}$.

(3) Finally, $\mathcal{S}$ responds $< j, U, \psi, \Omega_j) >$ to $\mathcal{A}$.

Challenge Phase

The challenger randomly selects a challenge $Chal = \{(i, v_i)\}_{i\in I}$, where $I = \{s_1, s_2, \cdots, s_c\}$, and a time period j^*. And then $\mathcal{S}$ provides $\mathcal{A}$ with $Chal$ and j^*. $\mathcal{S}$ requests $\mathcal{A}$ to prove that $\mathcal{A}$ possesses the blocks $\{m_{s_1}, \cdots, m_{s_c}\}$ of file $F = \{m_1, \cdots, m_n\}$.

Forgery phase $\mathcal{A}$ outputs a proof $P = (j, U, \sigma, \mu, \Omega_j)$ as the response proof. 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 $\mathcal{A}$'s proof $P = (j, U, \sigma, \mu, \Omega_j)$ is different from the real U in the set of authenticators $\Phi = (j, U, \{\sigma_i\}_{1\le i\le n}, \Omega_j)$ that $\mathcal{S}$ has kept, $\mathcal{S}$ abstracts tuple $< name||i||j^*, U, h, \lambda_i, * >$ from H_3 table to get $H_3(name||i||j^*, U) = g^{\lambda_i}$ with high probability. According to the analysis in APPENDIX A, $\mathcal{S}$ can solve the CDH problem in G_1 with high probability.

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

Game c'. *Game c'* is the same as *Game b'*, with only one difference. The challenger holds a list that contains its responses to the adversary's authenticator queries. The challenger observes each instance of the game with the adversary. If the adversary in any instance succeeds but the adversary's aggregate authenticator σ is not equal to the real aggregate authenticator $\sigma = \prod_{i\in I} \sigma_i^{v_i}$, then the challenger will abort.

Analysis. Let the file $F = \{m_1, \cdots, m_n\}$ with name $name$ causes the abort in time period j, and the corresponding set of authenticators provided by challenger are $\Phi = (j, U, \{\sigma_i\}_{1\le i\le n}, \Omega_j = (R_{\mathbf{w^j}|_1}, \cdots, R_{\mathbf{w^j}|_t}))$. Let $(j, Chal = \{(i, v_i)\}_{i\in I})$ be the query that causes the abort, and $P = (j, U, \sigma', \mu', \Omega_j)$ be the proof provided by the adversary. Let the expected response from an honest prover be $P = (j, U, \sigma, \mu, \Omega_j)$. The correctness of the proof means the following equation holds.

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

It is obvious that $\sigma \neq \sigma'$ and σ' can pass the verification equation because the challenger aborts. So

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

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

Here, we construct a simulator $\mathcal{S}$ to solve the CDH problem in G_1.

$\mathcal{S}$ is given as inputs tuple $(g, g^{a'}, v)$; his final goal is to compute $v^{a'}$. $\mathcal{S}$ behaves like the *Game b'* challenger, with the following differences:

In Setup phase, $\mathcal{S}$ sets $u = g^{\beta} v^{\gamma}$ for some $\beta, \gamma \in_R Z_q^*$. He also selects SK_0 by himself, and generates secret keys $S_{\mathbf{w^j}}$ of all the time periods j. $\mathcal{S}$ selects $\tau \in_R Z_q^*$, and sends $DK = \tau$ to $\mathcal{A}$.

$\mathcal{S}$ controls a random oracle H_3, and holds a list of queries. When $\mathcal{A}$ make a H_3 oracle query at $< name||i||j, U >$, $\mathcal{S}$ will check whether it is in a tuple $< name||i||j, U, h, \lambda, \gamma >$ of H_3. If it is in, $\mathcal{S}$ returns h to $\mathcal{A}$; Otherwise, selects $\lambda \in_R Z_q^*$, computes $h = g^{\lambda}$, and adds tuple $< name||i||j, U, h, \lambda, * >$ to H_3 table. Finally, $\mathcal{S}$ returns h to $\mathcal{A}$.

$\mathcal{S}$ holds a list of authenticator queries. When $\mathcal{A}$ queries the authenticator of block m_i with name *name* in time period j, $\mathcal{S}$ does as follows. He selects $\lambda_i, \eta \in_R Z_q^*$, computes $U = (g^{a'})^{\eta}$, and sets $h = g^{\lambda_i}/(g^{\beta} v^{\gamma})^{m_i}$. The probability that $H_3(name||i||j, U)$ has been defined is negligible. $\mathcal{S}$ can compute authenticator σ_i and respond it to $\mathcal{A}$ because $\sigma_i = H_3(name||i||j, U)^{a'\eta} \cdot S_{\mathbf{w^j}} \cdot u^{a'\eta m_i} = (g^{\lambda_i}/(g^{\beta}v^{\gamma})^{m_i})^{a'\eta} \cdot S_{\mathbf{w^j}} \cdot ((g^{\beta}v^{\gamma})^{m_i})^{a'\eta} = (g^{a'})^{\lambda_i \eta} \cdot S_{\mathbf{w^j}}$. $\mathcal{S}$ adds $< name||i||j, U, h, \lambda_i, \eta >$ to H_3 table.

If $\mathcal{A}$ responds with a proof $P = (j, U, \sigma', \mu', \Omega_j)$ in which σ' is different from the expected σ, and succeeds in the game, $\mathcal{S}$ extracts an tuple $< name||i||j, U, h, \lambda_i, \eta >$ from H_3 table. And then $\mathcal{S}$ divides 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 $\hat{e}(\sigma' \cdot \sigma^{-1} \cdot U^{-\beta\Delta\mu}, g) = \hat{e}((g^{a'})^{\eta}, v)^{\gamma\Delta\mu}$. So $\mathcal{S}$ can solve the CDH problem $v^{a'} = (\sigma' \cdot \sigma^{-1} \cdot U^{-\beta\Delta\mu})^{\frac{1}{\eta\gamma\Delta\mu}}$.

Therefore, we can construct a simulator $\mathcal{S}$ to solve the CDH problem in G_1 if there exists a non-negligible difference between the adversary's probabilities of success in *Game b'* and *Game c'*.

Game d'. *Game d'* is the same as *Game c'*, with only one difference. The challenger observes each instance of the game with the adversary. If the adversary succeeds in any instance but there is one adversary's aggregate message μ is not equal to the real aggregate message $\mu = \sum_{i\in I} v_i m_i$, then the challenger will abort.

According to the analysis in APPENDIX A, there exists a non-negligible difference between the adversary's probabilities of success in *Game c'* and *Game d'* as long as the CDH problem in G_1 is difficult.

As a result, there is only negligible difference probability between these games.

And then, we can construct a knowledge extractor to extract all challenged file blocks $m_{s_1}, \cdots, m_{s_c}$. By using independent coefficients $v_1, \cdots, v_c$ to execute Challenge phase on the same blocks $m_{s_1}, \cdots, m_{s_c}$ for c times, the extractor obtains c independent linear equations in the variables $m_{s_1}, \cdots, m_{s_c}$. It is easy for the extractor to extract $m_{s_1}, \cdots, m_{s_c}$ by solving these equations.

APPENDIX C
PROOF OF THE LEMMA 3

Proof. We can easily complete the proof based on Theorem 2 in [23]. According to Theorem 2 in [23], whenever an adversary $\mathcal{A}'$ in the security game (named as Game YRMV) of [23] that can cause the challenger to accept its proof with non-negligible probability, there exists an efficient knowledge extractor ε' that can extract the challenged file blocks except possibly with negligible probability. Let $\mathcal{A}$ be an adversary to attack Game 3, we will construct an algorithm $\mathcal{A}'$ against the security in [23]. $\mathcal{A}$ runs in the following phases.

1)**Setup phase**. $\mathcal{A}'$ goes into the Setup phase of Game YRMV to get the public key $(G_1, G_2, \hat{e}, g, u, T, H_1, H_2, H_3, R)$ and sends it to $\mathcal{A}$. $\mathcal{A}'$ also selects $DK = \tau \in_R Z_q^*$. Then $\mathcal{A}'$ computes $G = g^{\tau}$ and sends G to $\mathcal{A}$.

2)**Query phase**. $\mathcal{A}'$ selects to go into the Query phase of Game YRMV. When $\mathcal{A}$ queries the authenticator of any block m_i with name *name* in time period j, $\mathcal{A}'$ executes the authenticator query for m_i with name *name* in time period j in Game YRMV. Finally, $\mathcal{A}'$ sends the obtained authenticator to $\mathcal{A}$ as the reply. When $\mathcal{A}$ comes into time period $j+1$, $\mathcal{A}'$ also goes into this time period.

3)**Break-in phase**. When $\mathcal{A}$ comes into this phase in time period b, $\mathcal{A}'$ also goes into the Break-in phase of Game YRMV. Therefore, $\mathcal{A}'$ can get the current secret key SK_b. $\mathcal{A}'$ can easily compute ESK_b by the decryption key DK he holds. At last, $\mathcal{A}'$ provides $\mathcal{A}$ with ESK_b and DK.

4)**Challenge phase**. When $\mathcal{A}'$ is given a $Chal$ and a time period $j^*(j^* < b)$ in the Challenge phase of Game YRMV, he gives $\mathcal{A}$ the same challenge and time period.

5)**Forgery phase**. If $\mathcal{A}$ succeeds in providing a valid proof P to $\mathcal{A}'$ with non-negligible probability, $\mathcal{A}'$ outputs P as the proof in the Forgery phase of Game YRMV with the same probability. It is clear that this proof must be valid in Game YRMV.

According to Theorem 2 of [23], there must exist a knowledge extractor ε' that can extract the challenged file blocks except possibly with negligible probability. Note that the challenged blocks and time period are the same for $\mathcal{A}$ and $\mathcal{A}'$. So this theorem holds.

REFERENCES

[1] M. J. Atallah, K. Pantazopoulos, J. R. Rice, and E. E. Spafford, "Secure outsourcing of scientific computations," *Trends in Software Engineering*, vol. 54, pp. 215-272 2002.

[2] D. Benjamin and M. J. Atallah, "Private and cheating-free outsourcing of algebraic computations," *Proc. Sixth Annual Conference on Privacy, Security and Trust*, pp. 240-245, 2008.

[3] C.Wang, K. Ren, and J.Wang, "Secure and practical outsourcing of linear programming in cloud computing," *IEEE INFOCOM 2011*, pp. 820-828, 2011.

[4] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New Algorithms for Secure Outsourcing of Modular Exponentiations," *Proc. 17th European Symposium on Research in Computer Security*, pp. 541-556, 2012.

[5] 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.

[6] 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.

[7] H. Shacham and B. Waters, "Compact Proofs of Retrievability," *Advances in Cryptology-Asiacrypt'08*, pp. 90-107, 2008.

[8] 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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] B. Wang, B. Li and H. Li. Oruta, "Privacy-Preserving Public Auditing for Shared Data in the Cloud," *IEEE Transactions on Cloud Computing*, Vol.2, pp. 43-56, 2014.

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

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

[22] J. Yuan and S. Yu, "Public Integrity Auditing for Dynamic Data Sharing With Multiuser Modification," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1717-1726, Aug. 2015.

[23] J. Yu, K. Ren, C. Wang, V. Varadharajan, "Enabling Cloud Storage Auditing with Key-Exposure Resistance," *IEEE Transactions on Information Forensics and Security*. vol. 10, no. 6, pp. 1167-1179, Jun. 2015.

[24] D. Chaum and T. Pedersen, "Wallet databases with observers," *Advances in Cryptology-Crypto 1992*, pp.89-105, 1993.

[25] B. Chevallier-Mames, J. Coron, N. McCullagh, D. Naccache, and M. Scott, "Secure delegation of elliptic-curve pairing," *Proc. CARDIS 2010*, pp.24-35, 2010.

[26] S. Hohenberger and A. Lysyanskaya, "How to securely outsource cryptographic computations," *TCC 2005*, pp.264-282, 2005.

[27] M.J. Atallah and J. Li, "Secure outsourcing of sequence comparisons," *International Journal of Information Security*, pp.277-287, 2005.

[28] M.J. Atallah, and K.B. Frikken, "Securely outsourcing linear algebra computations," *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pp.48-59, 2010.

[29] X. Chen, J. Li, X. Huang, et al., "Secure Outsourced Attribute-based Signatures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3285-3294, 2014.

[30] F. Zhang, X. Ma, S. Liu, "Efficient computation outsourcing for inverting a class of homomorphic functions," *Information Sciences*, vol. 286, pp. 19-28, 2014.

[31] B. Lynn, The pairing-based cryptographic library, online at http://crypto.Stanford.edu/pbc/, 2015.