A COMPARATIVE STUDY IN VERIFIABLE OUTSOURCING OF KEY UPDATES FOR ENABLING CLOUD STORAGE AUDITING

Y. SHIVA KUMAR
B. Tech, Krishna Chaitanya Institute of Technology & Sciences,
Markapur, Prakasam (dist), A.P

B.V.SRINIVASULU
M. Tech, (Ph. D), Assistant professor
Krishna Chaitanya Institute of Technology & Sciences,
Markapur, Prakasam(dist),A.P
Bysanisrinu13@gmail.com

ABSTRACT:

Key-exposure resistance has always been animportant 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. In particular, 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 the 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, cloudstorage auditing, key update, verifiability.*

1.0 INTRODUCTION

Cloud computing, as a new technology paradigm with promising further, is becoming more and more popular nowadays. It can provide users with seemingly unlimited com-puting 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 seemingly 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, the privacy protection of data, the privacy protection of identities, dynamic data operations, the data sharing, 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. Yu *et al.* 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.

A.Related Work

Outsourcing Computation: How to effectively outsourcetime-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 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 out-sourcing of some scientific computations was proposed by Atallah*et al* Chevallier-Mames*et al*. designed the first effective algorithm for secure delegation of elliptic-curve pairings based on an untrusted server. The first out-sourcing algorithm for modular exponentiations was proposed by Hohenberger and Lysyanskaya, which was based on the methods of pre computation and server-aided computation. Atallah and Li proposed a secure outsourcing algorithm to complete sequence comparisons. Chen *et al*. proposed new algorithms for secure outsourcing of modular exponentiations. Benjamin and Atallah researched on how to securely outsource the computation for linear algebra. Atallah and Frikken gave further improvement based on the weak secret hiding assumption. Wang *et al*. presented an efficient method for secure outsourcing of linear programming computation. Chen *et al*. proposed an outsourcing algorithm for attribute-based signatures computations. Zhang *et al*. proposed an efficient method for outsourcing a class of homomorphic functions.

Cloud Storage Auditing: How to check the integrity of thedata stored in cloud is a hot topic in cloud security. The notion of "provable data possession" (PDP) was firstly proposed by Ateniese *et al*. to ensure data possession at untrusted servers. The notion of "proof of retrievability" (PoR) was pro-posed by Juels*et al.* [6] to ensure both possession and retrievability of data at untrusted servers. Wang *et al*. 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. The auditing protocols supporting dynamic data operations were also proposed in and Yang and Jia proposed an auditing protocol supporting both the dynamic property and the privacy preserving prop-erty. The privacy preserving of the user's identity for shared data auditing was considered. The problem of user revocation in shared data auditing was considered. Yuan and Yu proposed a public auditing protocol for data sharing with multiuser modification. Sookhak*et al*. proposed a public cloud auditing protocol for securing big data

storage based on algebraic signature. Guan *et al*. proposed the first cloud storage auditing protocol based on indistinguishability obfuscation, which is especially useful for low-power cloud users. Yang *et al*. proposed a public auditing protocol for shared cloud data supporting both iden-tity privacy and identity traceability.

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. 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 willadd 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. he 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 for the client; the second is to update the encrypted secret keys of the client in each time period. The TPA can be considered as a party with powerful computational capability or a service in another independent cloud. Similar to, the whole lifetime of the files stored in cloud is divided into T+ 1 time periods (from 0-th to*T*th time periods). Eachfile 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 secureoutsourcing of key updates is composed by seven algorithms (*SysSetup*, *EkeyUpdate*, *VerESK*, *DecESK*, *AuthGen*, *Proof-Gen*,*ProofVerify*), shown below:

1) **Sys Setup:** the system setup algorithm is run by the client. It takes as input a security parameter *k* andthe total

number of time periods T , and generates an encrypted initial client's secret key $E\ S\ K_0$, a decryption key $D\ K$ and a public key $P\ K$. Finally, the client holds $D\ K$, and sends $E\ S\ K_0$to the TPA.

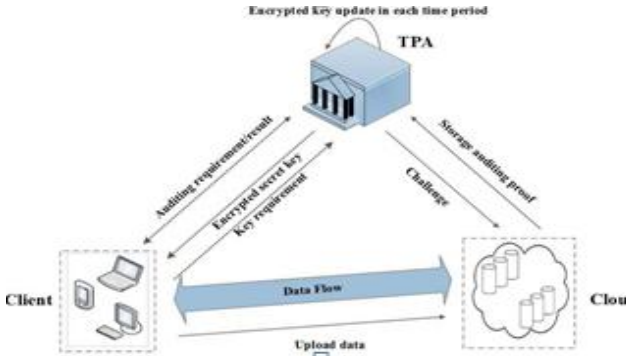

Fig. 1 System model of our cloud storage auditing.

Ekey Update: 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 P K, and generates a new encrypted secret key ESK j+1 for
period j + 1.

3) V er E SK: 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 P K,
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 P K, 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 P K, and generates the set of authenticators for F in time period j.

6) Proof Gen: the proof generation algorithm is run by the cloud. It takes as input a file F, a set of authenticators, a challenge Chal, a time period j and the public key P K, and generates a proof P which proves the cloud stores F correctly.

7) Proof V eri f y: 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 P K, and returns "True" if P is valid; or "False", otherwise.

Description of the Protocol

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*andthe total time period *T*.
   a) Then The client selects ρ,τ ∈R Z∗q , and computes R = gρ, G = gτ and E S = H1(R)ρ−τ .
   b) The client sets X0 = {(E S, R)} and 0 = ∅(where ∅ is null set), and sends the initial encrypted secret keys SK0 = (X0, 0) to the TPA. The client sets DK = τ , and keeps it himself. The client randomly selects a generator u of group G1.
   c) The public key is $P\ K=(R,G,u)$. Delete all intermediate data.
2) **Algorithm EkeyUpdate**: Input an encrypted secretkey $E\ S\ K_j$, the current time period j , and the public key $P\ K$. Parse $E\ S\ K_j=(X_{j,j})$. X_j is organized as a stack which consists of (ES_wj,R_wj) 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_wj,R_wj). Firstly, pop (ES_wj,R_wj) off the stack. Then do as follows:

If wj is an internal node (wj+1 = wj0 in this case), select ρw j 0, ρw j 1 ∈R Z∗q . And then compute Rw j 0 = gρw j 0 , Rw j 1 = gρw j 1, hw j 0 = H2(wj0, Rw j 0), hw j 1 = H2(wj1, Rw j 1), E Sw j 0 = E Sw j · H1(R)ρw j 0hw j 0 and E Sw j 1 =E Sw j · H1(R) ρw j 1hw j 1 . Push (E Sw j 1, Rw j 1) and (E Sw j 0, Rw j 0) onto the stack

orderly. Let X j+1 denote the current stack and define j+1 =j {Rw j 0}.

b) If wj is a leaf, define X j+1 with the current stack.

i) If wt = 0 (the node wj+1 is the right sibling node of wj in this case), then set j+1 ={Rw j+1 }−{Rw j} (Rw j+1 can be read from the new top (E Sw j+1 , Rw j+1) of the stack).

ii) If wt = 1 (wj+1 = w1 in this case, where w" is the longest string such that w0 is a prefix of wj), then set j+1 = j {Rw j+1 }− {Rw0, Rw01,⋯ , Rwt} (Rw j+1 can be read from the new top (E Sw j+1 , Rw j+1) of the stack).

c) Finally, erase key pair (E Sw j , Rw j), and return ESK j+1 = (X j+1, j+1).

3) **Algorithm VerESK**: Input a client's encrypted secretkey $E\ S\ K_j=(X_j,_j)$, the current period j and the public key $P\ K$. Verify whether the following equation holds:

$e\hat{}(g,\ E\ S_w\ j\ )= e\hat{}(\ R/\ t\ ^{h}{}_{w}\ j,\ H_1($ $G\cdot\ {}_{m1}{}^{R}w^{j}1\ {}_{1}\ R)),$ =

where$h_wj=H_2(w^j,R_wj)$.

If above equation holds, return 1; otherwise, return 0.

4) **Algorithm DecESK**: Input an encrypted client's secretkey $E\ S\ K_j$, a decryption key $D\ K$, the current period j , and the public key $P\ K$. The client decrypts the secret key as follows.

$S_wj\ = E\ S_w\ j\ \cdot H_1(\ R)^{\tau}$.

The real secret key is $S\ K_j=(X_j,_j)$, where X_j is the same stack as X_j except that the top element in X_j is

$(S_wj\ ,R_wj\ )$ instead of $(\ E\ S_wj\ ,\ R_wj\ )$ inX_j.

5) **Algorithm AuthGen**: Input a file$F=\{m_1,\cdots,m_n\}$,a client's secret key $S\ K_j$, the current period j and the public key $P\ K$.

a) The client parses $S\ K_j=(X_j,_j)$ and reads the top element (S_wj,R_wj) from the stack X_j . The client selects r ∈R Z∗q , and computes U = grbband σi = H3(name||i|| j, U)r · Sw j · urmi (i = 1,⋯ , 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 $=(j,U,\{\sigma_i\}_{1\le i\le n,j})$.

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$=(j,U,\{\sigma_i\}_{1\le i\le n,j})$, 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 $P\ K$ It then sends P= $(j,U,\ \sigma,\ \mu_{,j})$along with the file tag as theresponse 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 $P\ K$.

The TPA parses $_j=(R_wj_{|1},\cdots,R_wj_{|t})$. He then verifies the integrity of *name* and j by checking the file tag. After that, the client verifies whether the following where$h_wj=H_2(w^j,R_wj)$.If it holds, returns "$T\ r\ ue$", otherwise returns "$F\ alse$".

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 $V\ er\ E\ K\ ey$ 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 .

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 $P=(j, U, \sigma, \mu_j)$, theProofVerify algorithm always returns "*True*".

Proof: It is because the following equations hold:

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

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 bededuced from the following three lemmas.

Lemma 1: If the CDH problem in $G_1$is hard, wheneveran adversary *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, wheneveran adversary *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, wheneveran adversary *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): Ourauditing protocol is $\binom{t}{n}, 1-\left(\frac{n-1}{n}\right)^{c}$ 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. Thenumber 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 .

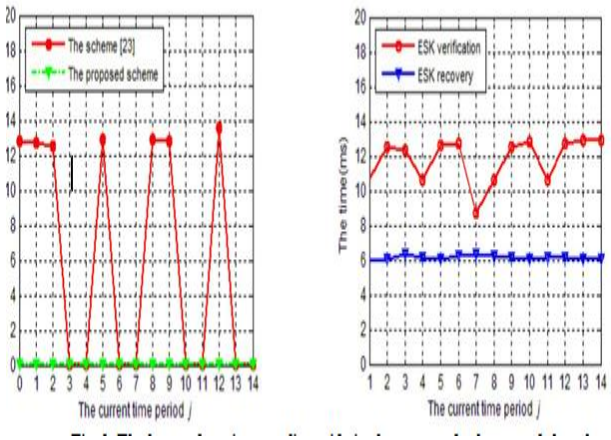

Fig. 3.The key update time on client side in the proposed scheme and the scheme [23].\

We have:

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

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

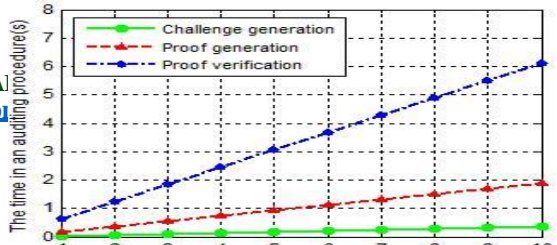

Fig. 5.The time of auditing procedures with different number of checked blocks.

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 [36]. 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 period T 14 which means we can use a full binary tree with depth 2 to associate with these time periods. All exper-imental results are taken as the mean values from multiple executions.

In the proposed scheme, the key update workload is out-sourced 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.

APPENDIX A

PROOF OF THE LEMMA 1

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

Game a'. *Game a'* is the same as Game 2, with only onedifference. 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 *S Si g* algorithm but not signed by the challenger.

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

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

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

Setup Phase: Firstly,$S$is given a tuple$(g, I_1= H_1(R)=\in G_1, I_2= R =g^b\in G_1)$. The aim of$S$is to compute^b , where where$b=\rho$ and $a,b\in Z_q^*$ are unknown to $*$, and computes$G=g^{\tau^{gaga}}$ and$A$.Sselects$\tau,\beta\in R$andZq $u =g^{\beta}$. Then$S$provides $P K =(R, G, u)$and $D K =\tau$to A.*Query Phase:* In order to answer the queries from$A$,Sneeds to make some preparation for key updates. We use $\mathbf{w^j}=w_1\cdot\cdot\cdot w_t$to denote the node associated with the currenttime period $j(j<b)$. He does as follows.

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

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

H_3 Queries: *S*holds a list of queries. When*A*queries H_3oracle at a point $<name||i||j,U>$, *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, *S* returns *h* to *A*.

(2) Otherwise, *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 *A*.

Authenticator Queries: *S*holds a list of queries. When*A*makes the authenticator query on block m_i with name *name* in time period *j* ,*S* does as follows.

(1) Firstly, *S* selects γ ,$\lambda_i \in_R Z_q^*$, 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 *S* is negligible. *S* adds $<name||i||j,U,h, \lambda_i, \gamma >$ to H_3 table.

If *U* in *A*'s proof $P=(j,U, \sigma, \mu,_j)$ is different from the real *U* in the set of authenticators $=(j,U,\{\sigma_i\}_{1\le i\le n},_j)$ that*S* has kept, *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, *S* can solve the CDH problem in$G_1$with highprobability.

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

Game c'.*Game c'* is the same as *Game b'*, with onlyone difference. The challenger holds a list that contains its responses to the adversary's authenticator queries. The chal-lenger 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 $j,Chal= \{(i, v_i)\}_{i\in I}$)be the query that causes the abort,and $P=(j,U, \sigma , \mu ,_j)$ be the proof provided by the adversary. Let the expected response from an honest prover be $P=(j,U, \sigma, \mu,_j)$. The correctness of the proof means the following equation holds.

It is obvious that $\sigma=\sigma$ 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 $\mu=\mu$, otherwise, it means $\sigma=\sigma$, which contradicts the above assumption. Let $\mu=\mu-\mu$.

Here, we construct a simulator *S* to solve the CDH problem in *G*1. is given as inputs tuple (g,g^a, v); his final goal is to compute v^a . *S* behaves like the *Game b'* challenger, with the following differences:

In Setup phase, S sets u = gβvγ for some β, γ ∈R Z∗q. He also selects SK0 by himself, and generates secret keys Swj of all the time periods j. S selects $\tau \in_R Z_q^*$ *, and sends DK = τ to A.*$S_{\mathbf{wj}}$ of all the time periods*j* .*S* selects$\tau \in_R Z_q^*$, and sends *D K* =$\tau$to*A*. S controls a random oracle H_3, and holds a list of queries. When *A* make a H_3 oracle query at $<name||i||j,U>$, *S* will check whether it is in a tuple $<name||i||j,U,h, \lambda, \gamma >$ of H_3. If it is in, *S* returns *h* to *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, *S* returns *h* to *A*.

S holds a list of authenticator queries. When *A* queriesthe authenticator of block m_i with name *name* in time period *j* ,*S* does as follows. He selects $\lambda_i, \eta \in_R Z_q^*$, computes$U = (g^a)^{\eta}$, and sets $h = g^{\lambda}i/(g^{\beta}v^{\gamma})^m i$.

$(g^{\lambda}i/(g^{\beta}v^{\gamma})^m i)^{a\eta} \cdot S_{\mathbf{wj}} \cdot ((g^{\beta}v^{\gamma})^m i)^{a\eta} = (g^a)^{\lambda}i^{\eta} \cdot S_{\mathbf{wj}}$.Sadds $< name||i \,||\, j, U, h, \lambda_i, \eta >$to H_3table.

If *A* responds with a proof $P=(j,U, \sigma , \mu ,_j)$ in whichσ is different from the expected σ , and succeeds in the game, *S* extracts an tuple $<name||i||j,U,h, \lambda_i, \eta >$ from H_3table. And then*S*divides the verification equation.

Therefore, we can construct a simulator 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 onlyone 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 $\mu=_{i\in I}v_i m_i$, 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 $m_{s1},\cdot\ \cdot\ \cdot,m_{sc}$. By using independent coefficients $v_1,\cdot\ \cdot\ \cdot,v_c$ to execute Challenge phase on the same blocks $m_{s1},\cdot\ \cdot\ \cdot,m_{sc}$ for c times, the extractor obtains independent linear equations in the variables $m_{s1},\cdot\ \cdot\ \cdot,m_{sc}$. It is easy for the extractor to extract $m_{s1},\cdot\ \cdot\ \cdot,m_{sc}$ by solving these equations.

APPENDIX C
PROOF OF THE LEMMA 3

Proof: We can easily complete the proof basedon [23, Th. 2]. According to [23, Th. 2], whenever an adversary A in the security game (named as Game YRWV) of [23] that can cause the challenger to accept its proof with non-negligible probability, there exists an efficient knowl-edge extractor ε that can extract the challenged file blocks except possibly with negligible probability. Let A be an adversary to attack Game 3, we will construct an algorithm A against the security in. A runs in the following phase.

1) **Setup phase.** A goes into the Setup phase of Game YRWV to get the public key (G1, G2, eˆ, g, u, T, H1, H2, H3, R) and sends it to A. A also selects$D$ $K=\tau\in_R Z_q^*$. Then A computes$G=g^{\tau}$and sends G to A.

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

3) **Break-in phase**. When*A*comes into this phase in timeperiod b, A also goes into the Break-in phase of Game YRWV. Therefore, A can get the current secret key S K_b. A can easily compute$E$ S $K_b$by the decryption key$D$ Kholds. At last, A provides A with E S K_b and D K .

4) **Challenge phase**. When*A*is given a*Chal*and a timeperiod $j^*(j^*<b)$ in the Challenge phase of Game YRWV, he gives A the same challenge and time period.

5) **Forgery phase**. If*A*succeeds in providing a validproof P to A with non-negligible probability, A outputs P as the proof in the Forgery phase of Game YRWV with the same probability. It is clear that this proof must be valid in Game YRWV.

According to [23, Th. 2], 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 A and A. So this theorem holds.

REFERENCES

[1] *M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. E. Spafford, "Secure outsourcing of scientific computations," Adv. Comput., vol. 54,215–272, 2002.*

[2] *D. Benjamin and M. J. Atallah, "Private and cheating-free outsourcing of algebraic computations," in Proc. 6th Annu. Conf. Privacy, Secur.Trust, 2008, pp. 240–245.*

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

[4] *X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," in Proc. 17th Eur. Symp. Res.Comput. Secur., 2012, pp. 541–556.*

[5] *G. Atenieseet al., "Provable data possession at untrusted stores," in Proc. 14th ACM Conf. Comput. Commun. Secur., 2007, pp. 598–609.*

[6] *A. Juels and B. S. Kaliski, Jr., "PORs: Proofs of retrievability for large files," in Proc. 14th ACM Conf. Comput. Commun. Secur., 2007,584–597.*

[7] *H. Shacham and B. Waters, "Compact proofs of retrievability," in Advances in Cryptology. Berlin, Germany: Springer-Verlag, 2008,90–107.*

[8] *G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in Proc. 4th Int. Conf. Secur. PrivacyCommun. Netw., 2008, Art. ID 9.*

[9] *F. Sebe, J. Domingo-Ferrer, A. Martinez-balleste, Y. Deswarte, and J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," IEEE Trans. Knowl. Data Eng., vol. 20, no. 8, pp. 1034–1038, Aug. 2008.*

[10] *R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in Proc. 28th IEEE Int. Conf. Distrib.Comput. Syst., Jun. 2008, pp. 411–420.*