Auditing Cloud Storage by Key Coverage Conflict in Cloud

Abhiram Srinivasan, Bharat Mallala, Sree Varun

Abstract— Maintaining data reliability in public cloud acts a vital role in cloud computing. Cloud storage auditing solves the complexity of data reliability in public cloud. In progress auditing protocols are all standard on the statement that the client's private key for auditing is completely protected. However, such statement probably will not for all time be seized, because of the probably feeble logic of protection and/or low protection settings at the consumer. If such a secret key for auditing is uncovered, nearly every one of the existing auditing protocols would certainly develop into incapable in the direction of exertion. In this paper, we meeting point happening this new fraction of cloud storage auditing. We examine how to decrease injure of the client's key coverage in cloud storage auditing, and provide the primary sensible solution for this original difficulty setting. We celebrate the meaning and the refuge model of auditing protocol with key- coverage flexibility and suggest such a protocol. In our plan, we utilize the preorder traversal technique and the binary tree structure to inform the private keys for the consumer. In addition to expand a novel authenticator structure to sustain the onward security and the assets of chunk less verifiability. The refuge proof and the presentation examination show that our proposed protocol is safe and proficient.

Keywords – key exposure, conflict, complexity, Authentication

I. INTRODUCTION

AUDITING cloud storage is used to authenticate the honesty of the data stored in public cloud, which is one of the significant protection techniques in cloud storage. In latest years, auditing protocols for cloud storage have concerned much concentration and have been researched exhaustively. These protocols focus on a number of dissimilar facets of auditing, and how to accomplish high bandwidth and computation effectiveness is one of the important distresses. For that purpose, the Homomorphic Linear Authenticator (HLA) technique that supports block less verification is explored to reduce the overheads of computation and communication in auditing protocols, which allows the auditor to authenticate the honesty of the data in cloud without retrieving the whole data. Many cloud storage auditing protocols like have been proposed based on this technique. The privacy protection of data is also an important aspect of cloud storage auditing. In order to reduce the computational trouble of the client, a third-party auditor (TPA) is introduced to help the client to periodically check the integrity of the data in cloud. However, it is possible for the TPA to get the client's data after it executes the auditing protocol multiple times. Auditing protocols are designed to ensure the privacy of the client's data in cloud. Another aspect having been addressed in cloud storage auditing is how to support data dynamic operations. Encompass proposed an auditing protocol maintaining entirely dynamic data operations including modification, insertion and deletion. Auditing protocols know how to also sustain dynamic facts operations. Additional aspects, such as alternate auditing, user revocation and eliminating credential management in cloud storage auditing have also been considered. However many investigate workings on cloud storage auditing have been done in recent years, a dangerous refuge problem. For some reasons, the user's secret key for cloud storage auditing will be disclosed to cloud. Initially managing key is a full difficult process, because some various elements are included like user training, system policy and so on. One client needs to manage lot of keys for finish various security problems. But these are very sensitive tasks. Because, if any mistakes are happened by clients, it will be going to be a big issue like the keys are disclosed to cloud. But it is a common one like, the client choose the cheap software tools to implement his security for reducing his economical factors. And then, the client may possible to attack by security attacks. Compared to standard organization, normal client cannot to provide a highly secured protection. And also there is a chance to ignore the protection to their system or unfortunately download the virus software and files from internet. Those cases are intended the hackers to hack their secret keys. Finally, cloud also has some benefits to give the known secret key to the hackers for storage auditing. So we need to aware on lot of situations. Each and every process should be very sensitive. Both security and cost effectiveness is the big issues in cloud computing. Particularly, if the keys are disclosed to cloud, it may forge their servers and reproduce the fraud data and import them into the servers. It will hide the data loss situations from servers. In a critical situations, it can even remove the user's data which are infrequently accessed to

save the space in storage area, even do not thinking about unsuccessful to give the auditing protocol started by the client. Surely, the secret key disclosing for the auditing process could be dangerous for the users of cloud storage applications. And knowing the way to handle the situation of disclosing secret key of the users is a very big issue. Our existing system did not consider those dangerous situations and also their technique is not auditing the keys correctly. In our system, we notice on how to decrease injure in the clients key disclosing in cloud data auditing. Our aim is to model a cloud storage auditing protocol along with the key exposure flexibility. Many of the new challenges are discussed below for new problems gives in many different deals. Initially, using the existing solutions of revocation keys to cloud is not the fact one. This is because, the client need to change the secret key and need to regenerate the previously stored data's authenticator when the client's key was exposed instead of auditing. The process contains the downloading of whole data from cloud, re-uploading the contents again to the cloud, generating new authenticators all of above can be boring ad awkward. In addition, it cannot assure that the real data of cloud provider when the client re-produces new authenticators. Then, directly adjusting typical key evolving process is also not the correct one for proposed problem setting. It can direct to downloading all of the actual files block when the authentication process is continued. This is partial, because this process is ill-assorted with block less authentication. The proposed authenticators leading to inappropriately high computation cannot be combined, and communication cost for the cloud storage auditing. Our contributions are summarized as follows.

1) We start the initial study on how to get the key-exposure flexibility in the cloud auditing protocol and starts new technique as protocol of auditing along with the key-exposure resiliencies. In that protocol, modifying or deleting client's data saved in cloud in earlier time periods can also found, even if the cloud acquire the user's present private key for cloud storage auditing. This very crucial issue is not notified before by existing auditing protocol designs. We further sanctify the security model and the definition of key-exposure resilience with auditing protocol for cloud storage auditing.

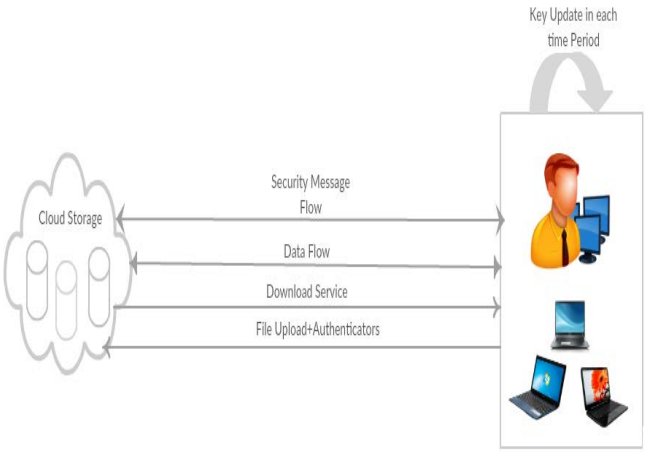

Fig 1. System Model

2) We analyze and model the initial useful auditing protocol with existing key-exposure resilience for cloud storage. To implement our aim, we utilize the structure of binary tree in an existing works on various cryptographic designs, to renew the private key of the client. This binary tree structure may be considered as a different of the tree structure used in the existing HIBE technique. And also, the pre-order traversal technique is utilized to combine each node of a binary tree in separate time period. In our proposed method, the stack structure is used to recognize the binary tree's pre-order traversal. We also model a novel authenticator sustaining the property of block less authenticity and the forward security.

3) We demonstrate our protocol's security power in the dignified security model, and validate its performance through concrete asymptotic analysis. In fact, the proposed system only combines realistic overhead to improve the key-exposure resilience. We also demonstrate that our proposed model can be improved to support the third party auditor, slothful update and various sectors.

The rest of the paper organized as follows: In section II, we present our system model, description, security model and preface of our work. Then, we provide concrete details of our protocol in section III. The security theorem and efficiency evaluation are given in section IV. Section V gives further discussions. We terminate the paper in section VI.

II. PROBLEM FORMULATION

A. System Model

We demonstrate an auditing system for secure authenticate cloud storage. The system includes two entities: the client (Data/Files owner) and the cloud. The client generates data, document and upload those data along with the corresponding authenticators to the cloud. The cloud stores these files for the user (client) and gives download service if the user wants. Each data file is furthermore split into multiple blocks. For the ease of description, we guess that the user also acts a vital role of auditor in our system because our system also supports for TPA. The user can frequently audit whether his data files in cloud are correct or not. The existence of data files stored in the cloud is split into t+1 time periods. In our system, the user will modify his private key for cloud storage auditing in the conclusion of each time period, but the public key is the static one. It cannot be modified. The cloud is permitted to get the user's private key for cloud storage auditing in one particular time period. That means the private key exposure can occur in this system model.

B. Security Model

a) The description of key exposure resilience for auditing protocol

Definition 1(Key exposure resilience for auditing protocol): Key exposure-resilience for auditing protocol is done by five algorithms (SysSetup, UpdateKey, AuthGen, ProofGen, and ProofVerify) these are explained in the following context.

1) SysSetup (1^k,T) → (PK,SK_0): This SysSetup algorithm is a probabilistic algorithm that takes input as time periods T, a security parameter k and creates the client's secret key SK_0 and public key PK. It is the client side process.
2) KeyUpdate(PK,i,SK_i) → (SK_{i+1}): Updating algorithm process for key is a probabilistic algorithm that takes input current time period i, client's secret key SK_i, and generate new secret key SK_{i+1} for the coming period i+1 and public key PK. This algorithm is client side process.
3) AuthGen(PK, i, SK_i , F*)* → (Φ*)*: this algorithm is a probabilistic algorithm which takes as input the current period *i* , the public key *PK*, a file *F*, and a client's secret key *SK i* creates the set of authenticators *Φ* for *F* in time period *i*. This algorithm is also client side process.
4) Proof Gen(PK, i, Challenge, F, Φ) → (P): this algorithm is a probabilistic algorithm which takes as input the public key *PK*, a challenge a file *F* , a time period i , and the set of authenticators *Φ*, and generates a proof *P* which means the cloud possesses *F*. This algorithm is also cloud side process.
5) Proof Verify(PK,i,Challenge, P) → ("True" or "False"): this algorithm is a deterministic algorithm which takes as input a time period i , the public key PK, a challenge and a proof P, and returns "True" or "False". This algorithm is run by the client.

b) Security Model

Our security model considers the concept of the data possession property and forward security. In Table I, we show a game to describe an adversary *A* beside the security of Key resilience process for auditing protocol. Exclusively, above game is composed of the following stages:

1) Setup Phase.

The client runs the SysSetup algorithm to generate initial client's secret key SK_0 and the public key PK. The client sends PK to an adversary and keeps SK_0 himself. Set time period i = 0.

2) Query Phase.

Adversary running in this phase can feasible to query as follows.

Authenticator Queries. Adversary can inquiry the authenticators of the chunks it chooses in time period i. It can flexible to choose a series of blocks $m_1 . . . m_n$, and sends them to the client. The client computes the authenticators for m_j (j = 1, . . . , n) in time period i , and sends them back to adversary. Adversary saves all blocks F = (m_1 . . . m_n) and their corresponding authenticators. Set time period i = i + 1. At the end of each time period, adversary can choose to still continue in query phase or move to the break-in phase.

Game Adversary-Forge (A)

i=0; A ← SysSetup (.);

Repeat

A← AuthGenSK$_i$ (F= (m_1….m_n));

SK$_{i+1}$ ← KeyUpdate (SK$_i$); i← i+1;

Until A goes to break-in phase;

b=i; A← SK$_b$;

A← (Challenge*(i*<b));

If ProofVerify (PK, i, Challenge, P) =1

Then return "True"

Else

Return "False".

3) Break-in Phase.

This phase models the possibility of key exposure. Set the break-in time period b=i. The client creates the private key (Secret key) SK$_b$ by the KeyUpdate algorithm and sends it to adversary. In Challenge Phase, the client sends a challenge and a time period i ∗(i ∗ < b). He also requests the adversary to provide a proof of possession for the blocks of file F = (m_1,. . .,m_n) under Challenge in time period j ∗, where 1 ≤ sl ≤ n, 1 ≤ l ≤ c, and 1 ≤ c ≤ n.

4) Forgery Phase.

Adversary outputs a proof for verifiability P for the blocks indicated by challenge in the period of time i*, and returns P. If function verify(PK, i*, Challenge, P)="True", after that adversary wins in that mentioned game. The described security model notes that an A(adversary) cannot cheat a valid proof for a particular period of time to key disclosure without owning all the block values corresponding to a provided challenge, if it cannot imagine all the missing bocks. The A (adversary) can be provide a private key for auditing in the key-disclosure in break-in time period. Evidently, the A (adversary) does not want to query the authenticators after or in the key-disclosure period of time because it can execute all private keys after this period of time using the uncovered secret key. The aim of the adversary (A) is to build a valid proof of ownership P for the blocks signified by challenge in the particular period of time i*. Definition 2 examines that there presents a knowledge Extractor permitting the extraction of the challenged block of files whenever adversary can generate a valid proof of ownership P in time period i*. Definition 3 denotes the finding ability for auditing protocol that verifies the cloud sustains the blocks that are not deal with high probability.

Definition 2 (Key Disclosure (Exposure) Resistance): We declare an auditing protocol is key exposure(disclosure) resistant if the coming condition keeps: when an adversary in above game that can cause the client to accept its verifications with non-negligible probability, there presents an efficient knowledge extractor, which can extract the challenged block of files apart from possibly with negligible probability.

Definition 3 (fundability): We declare an auditing protocol is (ϱ, δ) found able (0 < ϱ, δ < 1) if, given a fraction ϱ of corrupted blocks, the probability that the corrupted blocks are found is at least δ.

III. PROPOSED PROTOCOL

We initially examine two basic solutions for the key-disclosure problem for auditing the cloud storage before we provide our protocol. The starting one is naïve solution, which in fat cannot easily solve this problem. The following one is better solution than the before one, which can resolve this problem but has a huge overhead. They are both not practical when substituted in realistic settings. After that we provide our protocol that s more effective and efficient than the both existing solutions.

A. Naïve solution

In this method, the user uses the traditional key revocation method. Once the user knows his private key and the regarding public key. At the same time, user produces one newly generated public key and private key, and uses certificate update to disclose the newly generated public key. The authenticators of the data already saved in cloud,

Though, all want to be updated because the existing private key is no high secure. Thus, the user wants to download all his data stored form the cloud, give new authenticators for users using the new private key, and then save new authenticators to the cloud. Definitely, it is a difficult procedure, and uses lot of resources and time. Additionally, cloud storage auditing private key is already known to cloud and also the authenticators too known to cloud. It is very complex for the client to sure the authenticators and the accuracy of downloaded data from the cloud. Finally, reproducing public key and private key cannot fully solve the problem.

B. Slightly Better Solution

The client originally produces a series of public keys and secret keys: (PK1,SK1), (PK2,SK2), . . . , (PKT ,SKT). Let the permanent public key be (PK1, . . . , PKT) and the secret key in time period j be (SK j , . . . , SKT). If the client uploads records to the cloud in time period j , the client uses SK j to subtract authenticators for these records. Then the client uploads files and authenticators to the cloud. While auditing these files, the client uses PKj to verify whether the authenticators for these archive are really generated from end to end SK j . When the time period modify from j to j + 1, the client erase SK j from his storage. Then the new secret key is (SK j+1. . . SKT). This explanation is obviously better than the naive solution. Note that the keys SK1, . . . , SK j−1 have been deleted through or before time phase j . So from this period elapsed, the cloud cannot forge any authenticator uploaded in earlier time periods, even if the secret key (SK j , . . . , SKT) in time period j for inspection is uncovered. It means the obscure cannot change the client's data and forge any authenticator that can be established under PKt (t < j) whilst it get the client's secret key in time period j . Though, the drawback of this explanation is the following: the public key and the secret key have to be extremely extensive and linear with the total figure of possible time periods T, which is imaginary to well over the lifetime of information to be stored in the cloud. As the unremitting trend of immigration to cloud, it is not hard to envision the T value to be very large, making such linear overhead practically unacceptable.

C. *Our Cloud Storage Auditing With Key-Exposure Resilience*

Our goal is to propose a convenient auditing protocol with key-exposure resilience, in which the equipped complexity of key size, multiplication overhead and announcement overhead ought to be at most sub linear to T. In classify to achieve our goal, we use a binary tree structure to assign time periods

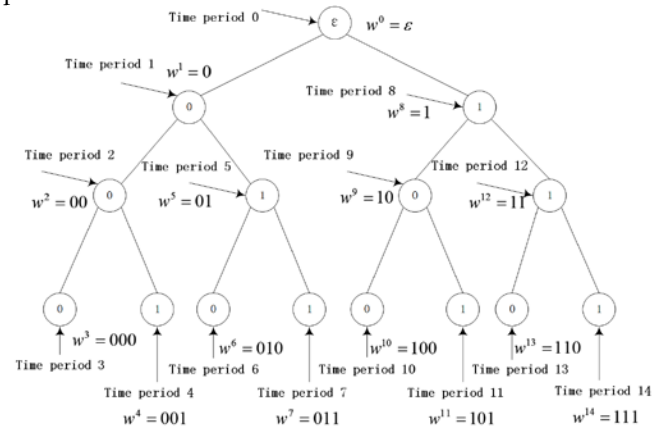

Fig. 2. An example of how to associate the nodes with time periods in a binary tree with depth 4.

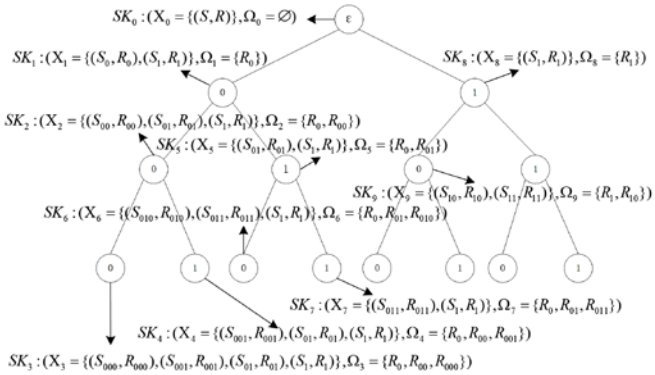

Fig. 3. An example to show what elements are included in SK j ($0 \le j \le 9$) when l = 4.

and connect periods with tree nodes by the pre-order traversal method [24]. The secret key in apiece time period is controlled as a stack. In each time period, the secret key is rationalized by a forward-secure method [28]. It guarantee that any authenticator produce in one time period cannot be compute beginning the secret keys for any extra time period later than this one. Besides, it helps to ensure that the difficulty of keys size, calculation overhead and announcement overhead are only logarithmic in total numeral of time periods T. As a result, the auditing procedures achieve key-exposure flexibility while rewarding our competence requirements. As we will show later, in our protocol, the client can audit the veracity of the cloud data still in comprehensive manner, i.e., without rescue the entire data starting the cloud. As same as the key-evolving machinery [21]–[23], our proposed procedure does not regard as the key exposure conflict during one time period. Below, we will give the meticulous depiction of our core practice.

D. *Description of Our Protocol:*

1) SysSetup: Input a security parameter k and the total time period T. Then

a) Run IG(1k) to generate two multiplicative groups G1, G2 of some prime order q and an admissible pairing ˆe : G1 × G1 → G2.

b) Choose cryptographic hash functions H1 : G1 → G1, H2 : {0, 1}∗ × G1 → Z∗q and H3 : {0, 1}∗ × G1 → G1. Select two independent generators g, u ∈ G1.

c) The client selects ϱ ∈ Z∗q at random, and computes R = gϱ and S = H1(R)ϱ .

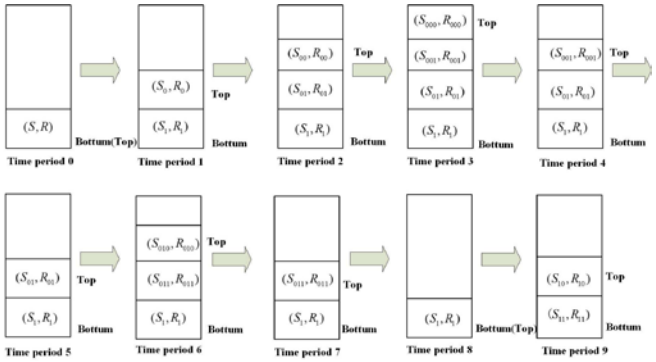

Fig. 4. An example to show the stack changes from time period 0 to time period 9 when l = 4.

d) The public key is PK = (G1, G2, ˆe, g, u, T, H1, H2, H3, R). Set X0 = {(S, R)} and 0 = ∅ (where ∅ is null set). The initial secret key is SK0 = (X0,0).

2) KeyUpdate: Input the public key PK, the current time period j and a secret key SK $_j$. Denote the node associated with period j with a binary string $w_j = w1 \cdots wt$.

As we have mentioned in this section, X j is organized as a stack which consists of (S_{wj} , Rwj) and the key pairs of the right siblings of the nodes on the path from the root to wj.

The top element of the stack is (S_{wj} , R_{wj}). Firstly, pop (S_{wj} , R_{wj}) off the stack. Then do as follows: a) If w_j is an internal node (Note w_{j+1} = wj0 in this case), then select ϱwj0, ϱwj1 ∈ Z∗q, and compute $R_{wj0} = g^{\varrho_{wj0}}$, $R_{wj1} = g^{\varrho_{wj1}}$,

$S_{wj0} = S_{wj} \cdot H1(R)^{\varrho_{wj0}} h_{wj0}$ and $S_{wj1} = S_{wj} \cdot H1(R)^{\varrho_{wj1}} h_{wj1}$, where $h_{wj0} = H2(w_j^0, R_{wj0})$ and $h_{wj1} = H2(w_j1, Rwj1)$. Push (Swj1, Rwj1) and (Swj0, Rwj0) onto the stack orderly. Let X j+1 denote the current stack and define j+1 = j {Rwj0}. b) If w_j is a leaf, define X j+1 with the current stack. i) If wt = 0 (Note that the node wj+1 is the right sibling node of wj in this case),

Then set j+1 = j {Rwj+1} − {Rwj } (Rwj+1 can be read from the new top (Swj+1 , Rwj+1) of the stack). ii) If wt = 1 (Note that 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 {Rwj+1} − {Rw0, Rw01, . . . , Rwt }. Finally, Return SK j+1 = (X j+1,j+1). We give an case to show the stack modify from time period 0 to time period 9 while l = 4 in Fig. 4. As shown is Fig. 3, the time periods j (j=3,4,6,7) correspond to the leaves of the binary tree in this example. So the KeyUpdate algorithm ought to run b and c steps in these time periods. While added time periods j (j=0,1,2,5,8,9) be in contact to the internal nodes of the binary tree. So the KeyUpdate algorithm should run a and c steps in these time periods. The changes of Ωj (0 ≤ j ≤ 9) are shown as follows.

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

TABLE II
EFFICIENCY COMPARISON

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

Here, (j, Challenge) pair is given by the auditor, and then used by the cloud. An aggregated authenticator is calculated by cloud Φ = (i, U, σ, Ω,j), where $\sigma = \prod_{i \in I} \sigma_i^{v_i} \cdot i$. It also calculates the linear combination of sampled blocks $\mu = \sum_{i \in I} v_i m_i$. It then sends the public key along with the file tag as the response proof of storage accuracy to the client.

5. Proof Verify: Input the time period j, challenges challenge, public key and a proof P. These are all notice the node, that is connected with the period i. The user parses the values. Then the user checks the integrity of time, name by verifying the file tag. Then, the user checks the following conditions.

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

If it doesn't holds, returns "False", otherwise returns "True".

IV. SECURITY AND PERFORMANCE

A. *Security analysis*

Theorem 1 (Accuracy): For each random challenge and one valid proof P, the ProofVerify algorithm always returns the value true.

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

B. *Performance analysis*

In Table II, we provide the effectiveness assessment between our protocol and existing protocol based on BLS signature method. We select existing protocol as a benchmark is mostly because the manufacture of it is generally analyzed as very professional. It is also the most associated to our manufacture. Here, T_e denotes the time costs of T_P and exponentiation on the group G1denotes the time costs of bilinear pairing from G2 to G1. Other processes like the multiplication on G1, set operations, stack operations, the operations on Zq and G2 and hashing operations are negated because they just donate omit table computation costs. Note that it is natural for our protocol to add more transparency than existing protocol in order to realize the extra key-exposure flexibility. Because we utilize the pre-order traversal technique and binary tree structure to connect the time periods and update secret keys, as shown in Table II, our protocol realizes nice performance. The costs of the KeyUpdate algorithm, SysSetup algorithm, the AuthGen algorithm, the Proof Gen algorithm are independent of the total number of time periods T.

TABLE III
COMPLEXITIES OF KEYS SIZE AND COMMUNICATION OVERHEADS

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

In Table III, we provide the complexity comparison of communication overhead and key size between our protocol and existing protocol. The public key size is independent of T and the private key size of the client is only logarithmic in T. It is much improved than the slightly better solution, in which the private key size and the public key size are both linear with T. The difficulty of the challenge overhead in our protocol and existing protocol are both O(k) (here k is the security parameter). The difficulty of the response overhead is O((logT)k) in our protocol because the response proof wants to hold the set of verification values X j . Usually speaking, the difficulties of communication overhead and key size in our protocol are at most logarithmic in T, which is completely suitable in practice.

V. CONCLUSION

In this paper, we revise on how to compact with the client's key coverage in cloud storage auditing. We suggest a new pattern called auditing protocol with key coverage flexibility. In such a protocol, the honesty of the data beforehand stored in cloud can unmoving be established still if the client's current secret key for cloud storage auditing is uncovered. We celebrate the classification and the safety model of auditing protocol with key-coverage flexibility, and after that suggest the first sensible solution. The safety proof and the asymptotic presentation estimation show that the projected protocol is protected and proficient.

REFERENCES

1] 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.

[2] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in Proc. 4th Int. Conf. Secur. Privacy Commun. Netw

[3] F. Sebe, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-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.

[4] 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.

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

[6] C. Wang, K. Ren, W. Lou, and J. Li, "Toward publicly auditable secure cloud data storage services," IEEE Netw., vol. 24, no. 4, pp. 19–24, Jul./Aug. 2010.

[7] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Efficient provable data possession for hybrid clouds," in Proc. 17th ACM Conf. Comput. Commun. Secur. 2010, pp. 756–758.

[8] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in Proc. 4th Int. Conf. Secur. Privacy Commun. Netw.

[9] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 9, pp. 1717–1726, Sep. 2013.

[10] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacypreserving public auditing for secure cloud storage," IEEE Trans. Comput., vol. 62, no. 2, pp. 362–375, Feb. 2013.

[11] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 5, pp. 847–859, May 2011.

[12] Y. Zhu, G.-J. Ahn, H. Hu, S. S. Yau, H. G. An, and C.-J. Hu, "Dynamic audit services for outsourced storages in clouds," IEEE Trans. Services Comput., vol. 6, no. 2, pp. 227–238, Apr./Jun. 2013.

[13] C. Erway, A. Küpçü, C. Papamanthou, and R.Tamassia, "Dynamic provable data possession," in Proc. 16th ACM Conf. Comput. Commun. Secur. 2009, pp. 213–222.

[14] H. Wang, "Proxy provable data possession in public clouds," IEEE Trans. Services Comput., vol. 6, no. 4, pp. 551–559, Oct./Dec. 2013.

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

[16] H. Wang, Q. Wu, B. Qin, and J. Domingo-Ferrer, "Identity-based remote data possession checking in public clouds," IET Inf. Secur., vol. 8, no. 2, pp. 114–121, Mar. 2014.

[17] T. Stewart. (Aug. 2012). Security Policy and Key Management: Centrally Manage Encryption Key. [Online]. Available: http://www.slideshare. Net/Tina-stewart/security-policy-and-enterprise-key-management-fromvormetric

[18] Microsoft. (2014). Key Management. [Online]. Available: http://technet.microsoft.com/en-us/library/cc961626.aspx

[19] FBI. (2012). Is Your Computer Infected with DNSChanger Malware?. [Online]. Available: http://www.fbi.gov/news/news_blog/is-yourcomputer-Infected-with-dnschanger-malware

[20] FBI. (2011). Botnet Operation Disabled. [Online]. Available: http://www.fbi.gov/news/stories/2011/april/botnet_041411

[21] M. Bellare and S. Miner, "A forward-secure digital signature scheme," in Advances in Cryptology—CRYPTO. Berlin, Germany: Springer-Verlag, 1999, pp. 431–448.

[22] Y. Dodis, J. Katz, S. Xu, and M. Yung, "Key-insulated public key cryptosystems," in Advances in Cryptology—EUROCRYPT. Berlin, Germany: Springer-Verlag, 2002, pp. 65–82.

[23] G. Itkis and L. Reyzin, "SiBIR: Signer-base intrusion-resilient signatures," in Advances in Cryptology—CRYPTO. Berlin, Germany: Springer-Verlag, 2002, pp. 499–514.

[24] R. Canetti, S. Halevi, and J. Katz, "A forward-secure public-key encryption scheme," in Advances in Cryptology—EUROCRYPT. Berlin, Germany: Springer-Verlag, 2003, pp. 255–271.

[25] F. Hu, C.-H. Wu, and J. D. Irwin, "A new forward secure signature scheme using bilinear maps," Cryptology ePrint Archive, Tech. Rep. 2003/188, 2003. [Online]. Available: http://eprint.iacr.org/2003/188