Received June 3, 2021, accepted June 28, 2021, date of publication July 2, 2021, date of current version July 12, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3094264

An Efficient Data Auditing Protocol With a Novel Sampling Verification Algorithm

XUELIAN LI[1], **LISHA CHEN**[1], **AND JUNTAO GAO**[2]
[1]School of Mathematics and Statistics, Xidian University, Xi'an, Shaanxi 710071, China
[2]Guangxi Key Laboratory of Cryptography and Information Security, School of Telecommunications Engineering, Xidian University, Xi'an, Shaanxi 710071, China

Corresponding author: Lisha Chen (lishachen25@163.com)

This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB0800601, in part by the Natural Science Foundation of China under Grant 61303217 and Grant 61502372, and in part by the Guangxi Key Laboratory of Cryptography and Information Security under Grant GCIS201802.

ABSTRACT Currently, data auditing is an important means to check the integrity of the data stored on the cloud. Existing data auditing schemes have done a lot of work in terms of functionality, implementation, and security. As we can see, all of these schemes are based on the auditing framework proposed by Ateniese *et al.*, which uses random sampling to probabilistically check whether the data is integrity or not. But multiple repeated sampling may lead to an intersection between the challenge sets selected each time, reducing the probability of detection, and the time for detecting corrupted blocks will be indefinite. If the corrupted data is not found for a long time, the data remedial measures will become invalid, resulting in permanent data loss. To address the above problem, we designed an efficient sampling verification algorithm. Based on this algorithm, the auditing scheme is further optimized, and a dynamic auditing function has been developed for the scheme to facilitate data owners to update data. Through the security analysis, our scheme is soundness. The experiment shows that our sampling verification can detect corrupted blocks faster than other auditing schemes.

INDEX TERMS Cloud storage, data auditing, dynamic structure, sampling verification, smart contract.

I. INTRODUCTION

Cloud storage is a popular way to store data online, which allows the data owner (DO) to efficiently access data at any time and anywhere. Especially during the COVID-19 crisis, the global cloud storage market is expected to grow from US$52 billion in 2020 to US$204.4 billion in 2027. The compound growth rate during 2020-2027 is as high as 21.6% [1]. Cloud storage has rapidly developed into a mainstream way for people to store data.

However, data stored in the cloud is at risk of data loss due to malicious deletions by the cloud service provider (CSP) to save storage space [2], [3]. Specifically, the CSP may delete some data that the DO rarely access and conceal the fact of data loss from DO to gain extra storage space and a good reputation. Since the amount of data stored in the cloud is very large, it is unrealistic for DOs with limited storage and computing power to detect the correctness of all the data blocks are stored one by one. Ateniese *et al.* start the first study on a novel auditing scheme, which detects the integrity through random sampling combined with homomorphic verifiable aggregate signature technology (HVA). With the gradual exhaustion of DO's storage and computing power, the third-party public auditor (TPA) accepted the appointment of DOs and became the computing power of DOs. Thus, the public auditing scheme was put forward [4], and its auditing process is shown in Fig. 1. First, the DO uploads the data to the cloud and deletes the local data. Second, the DO delegates audit tasks to TPA. Third, TPA and CSP regularly conduct challenge-proof interactions to detect the integrity of data stored in the cloud. The challenge-proof interaction is achieved by random sampling, which is the key to detection, and the process is that TPA randomly chooses c data blocks from the data file F divided into n data blocks for detection and sends the challenge set that the index of c data blocks $K_1 = \{s_1, s_2, \cdots, s_c\}_{s_j \in \{1,\cdots,n\}}$ to CSP. Upon receiving K_1, the CSP generates an HVA signature for the challenged data as a data possession proof and sends the proof

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Asif.

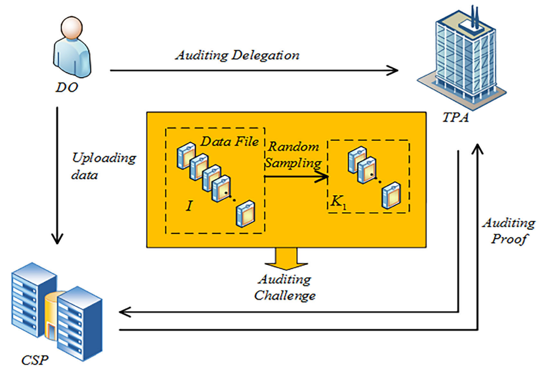


FIGURE 1. **Data auditing protocol.**

to TPA, the TPA verifies the validity of the signature to get the checking result.

However, when TPA starts to check the data regularly, one random sampling turns into multiple repeated random sampling, which makes the time to detect data corruption become an indefinite value. In most cases, it is impossible to detect it immediately, since the challenged data blocks set extracted each auditing may have an intersection, which reduces the efficiency of detection. Even more, the CSP may get away with deleting blocks of data that are rarely detected to save storage space. Once the data loss occurs over a period of time from being discovered by the DO. the DO is unable to take remedial measures right away, with the passage of time, the DO's remedial measures become ineffective, resulting in the permanent loss of data. It is undoubtedly a huge loss for DOs, who pay for storage services but get no data security in return.

In recent years, a large number of scholars have proposed functional, implementation, and security improvements to auditing schemes, but none of them mentioned this issue. As DOs's demands increased, data auditing scheme has developed many functions, such as identity privacy, data privacy preserved, user Revocation, batch auditing, dynamic data auditing, group management, etc [5]–[8]. Especially, for electronic medical record data, the sensitive information hiding function is useful [9]. Moreover, many scholars study how to resist possible attacks during the auditing process, such as key exposure attacks, replacement attacks, replay attacks, etc [10]–[13]. Furthermore, the scheme, which TPA acted by the working nodes in the smart contract helps the DO not need to pay service fees for lost data and avoids the collusion attack between TPA and CSP [14], [15], began to catch on. Currently, no one cares about how to improve the probability of successful detection and make the indefinite value certain.

But the indefinite value is like an uncertain bomb, which is always accompanied by the auditing scheme to give the adversary a chance to attack. In the latest combination of smart contract auditing schemes, the number of working nodes is very large, if each working node generates a challenge set, CSP needs to generate huge proofs for all the challenges, which will be a huge burden for CSP. So, the challenge set can only be generated by CSP. As described in [14], CSP selects the value of c and uses a random function to generate the challenge set. Whereas, since TPA uses repetitive random sampling in each auditing [16], [17], CSP can select the value of c to generate a challenge set that can pass the detection, thereby reducing the probability of successfully detecting a corrupted block, which may result in users losing their data and paying for illegal services. Our scheme allows users to choose appropriate c by themselves, and after reaching an agreement with CSP, writing c into the smart contract to avoid malicious modification by CSP. To increase the probability of successful detection, we design a novel sampling verification algorithm that each random sampling is performed in the data outside the previous challenge set, which is equivalent to select c data blocks in $(n - c)$ data blocks for testing in each auditing. Moreover, after sampling three times, all corrupted blocks will be gathered in the set to be sampled for the next time, which can increase the probability of detecting corrupted blocks.

Besides, the data stored on the cloud is not static. DOs may find that new data needs to be recorded or errors in the original data need to be corrected. Therefore, the updating of data version has become indispensable [18]–[20]. Erway *et al.* [21] proposed a dynamic provable data possession (DPDP) scheme, which introduces a rank-based authenticated skip list to realize the dynamic auditing, but it does not support public auditing. Wang *et al.* [22] presented a dynamic provable data possession (DPDP) scheme based on the Merkle-hash tree (MHT), which supports privacy protection and batch auditing. However, both of the above two schemes realize the dynamic data update operation at the cost of great communication overhead. Zhu *et al.* [23] designed a public auditing scheme (IHT-PA) based on the index-hash table (IHT), but the sequence structure of IHT caused the average $\frac{n}{2}$ elements to be readjusted during insertion and deletion operations (where n is the number of blocks in a file), and the signature of the data block whose sequence number was changed needs to be recalculated, which is very Inefficient. Tian *et al.* [24] implements a dynamic-hash table (DHT) based public auditing (DHT-PA) scheme. Although the communication overhead is reduced compared to the above schemes, the lack of index switchers in the scheme makes it impossible to quickly locate a specific data block. Shen *et al.* [25] presented a public auditing (DLIT-PA) scheme, which combines the doubly-linked information table with the location array, successfully solved the index switching problem, and can efficiently implement dynamic auditing and batch auditing functions, but scheme [25] could not support the privacy preserved. We also implement a dynamic auditing function to help DOs update data, which solves the privacy problem scheme [25] by using random mask technology, and the design of the non-interactive process makes the communication overhead only consumes constant level. The design of the ESV algorithm makes our scheme can avoid The emergence of inefficient audit situations.

A. OUR CONTRIBUTIONS

We design a secure auditing scheme, in which TPA acts by the working nodes in the smart contract to resist the collusion attack between CSP and TPA. Although our scheme is also based on the auditing framework proposed by Ateniese *et al.*, instead of using the traditional random sampling method, we design a novel sampling verification algorithm to improve the efficiency of the scheme. Our scheme guarantees that If the data stored on the cloud is lost, DO will not need to pay a service fee, on the contrary, CSP needs to pay compensation to DO. The main contributions of our efficient data auditing scheme are as follows:

1) In our scheme, the size of the challenge set should be chosen by DO in advance, and it will be written into the smart contract after reaching an agreement between DO and CSP, which can resist the replacement attack from CSP.
2) We are the first to consider how to improve the probability of successful detection. We designed a new sampling verification algorithm to ensure that there is no intersection between any adjacent challenge sets in two audits, which collecting corrupted blocks together on a small scale to increase the probability of successful detection. The experiment shows we can detect corrupted blocks faster than other auditing schemes.
3) We develop the functions of data privacy-preserving, dynamic auditing, and batch auditing. The security analysis proves that our data auditing scheme is sound. Through performance evaluation, we can see that our scheme can verify the integrity of data more effectively.

B. ORGANIZATION

The rest of this article is arranged as follows: In Section III, we describe the symbols, the background of the doubly linked info table and location array, and two verification algorithms used in this paper. In Section IV, we give the formal definition of a non-interactive public data possession scheme and systematically analyze its threat model and security model. The design details of our efficient sampling verification and auditing protocol will be respectively given in Section V and Section VI. In Section VII, we prove the soundness of our protocol with the game proof, and the performance evaluation of our auditing scheme and our sampling verification will be shown in Section VIII. Finally, we conclude this paper in Section IX.

II. RELATED WORK

With the increase in user demand for cloud storage services, many data auditing schemes have been proposed. With the popularity of the Internet of Things in people's lives, data is no longer static, and users need to update and maintain their data frequently to keep the data fresh. Reference [18] proposed a dynamic audit program, with the help of doubly linked information table, users can update their data to ensure that the updated data can still pass the data auditing verification. Reference [8] proposed to allow group members to cooperate in the dynamic data updating operation through group management. The group manager maintains the IKL list and IBL list. The data updated by the user will be recorded in the IBL list. Once the data is damaged, the group manager can find the user who recently operated the data by searching the IBL list, and query the IKL list to reveal its identity. But as mentioned in [5] and [19], the participation of manager or TPA may lead to abuse of authority or data loss, so how to ensure data privacy and resist collusion attacks becomes very important. Reference [5] proposed to use random masking technique to blind data ensuring data privacy against TPA. At the same time, shamir secret sharing is used to tracing the user's identity without group managers to resist collusion attacks. Reference [14] solves the above problem by allowing the working node of smart contract to act as a TPA. And it can support the pay-as-you-go payment model, which means that, when the CSP causes data loss, the DO no longer need to pay service fees, users only need to pay for the services they have enjoyed. Reference [9] uses a sanitizer to filter the sensitive information contained in the data block, and convert the signature of the sanitized file into a valid signature, which can hide the sensitive information in the file when the user shares the cloud file. But as mentioned in [15], the auditing scheme with sensitive information hiding for secure cloud storage has some defect. First, it lacks a fair arbitration mechanism, and secondly, CSP may store redundant and repeated data, leading to increased storage overhead. Reference [15] combines the smart contracts with the message-locked encryption algorithm to address the above problems, which ensures that malicious clouds will automatically pay fines to users who lose data as compensation.

III. PRELIMINARIES

A. NOTATIONS

The notations of our scheme is shown in Table. 1.

TABLE 1. **Notations.**

Notation	Description
DO	The data owner
CSP	The cloud service provider
TPA	The third-party Auditor
e	A bilinear pairing map
$H_1(\cdot), H_2(\cdot)$	A collusion-resistant hash function
$\mathcal{F}(\cdot)$	A pseudorandom function
$\mathbb{G}, \mathbb{G}_T$	Multiplicative groups with order p
g, u	Generators of group $\mathbb{G}$
$\mathbb{Z}_p$	Set of nonnegative integers less than p
τ	The current public status information
m_i	The general name of data blocks
n	The number of blocks in the original data file
I	The Subscript set of original data file, $I = \{1, 2, \cdots, n\}$
c	The number of challenged data blocks
k	The number of corrupted blocks
K_0	The challenge set of last auditing
K_1	The Subscript set of *chal*
Loc_i	Specific location of m_i

B. COMPLEXITY ASSUMPTION

Definition 1 (Computational Diffie-Hellman Problem): Given $(g, g^x, h) \in G$, where g is the generator of $\mathbb{G}$ and $x \in_R \mathbb{Z}_p$, computes h^x.

Definition 2 (Bilinear Pairing): Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative groups of the prime order p, and g_1, g_2 be generators of group $\mathbb{G}_1$.A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with following properties:

1) Bilinear: for all $g_1, g_2 \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, we can get $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
2) Non-degeneracy: $e(g_1, g_2) \neq 1$.
3) Computability: there is an efficient algorithm to compute this pairing.

C. DOUBLY LINKED INFORMATION TABLE (DLIT) AND LOCATION ARRAY (LA)

As defined in [9] and [25], DLIT is a two-dimensional data structure used to record data information shown in Fig. 2. The data information includes two aspects: the file information and the block information. The file information is written on the left, which contains the user's identity U_{ID} and the file's identity F_{ID}, since a user may have multiple files stored on the cloud. The right one is the block information, which contains the version number $V_{v,n}$ and a timestamp $T_{u,f,n}$. The version number and the time stamp are corresponded to the n_{th} block of the f_{th} file of the u_{th} user. Both file information and block information use a doubly linked data structure, in detail, linking the next information record forward, and linking the previous information record backward. Location Array (LA) switches the index to the specific location of data blocks. The element m_i in $l_{x,y}$ of the array means the block name of the y_{th} block of x_{th} file.

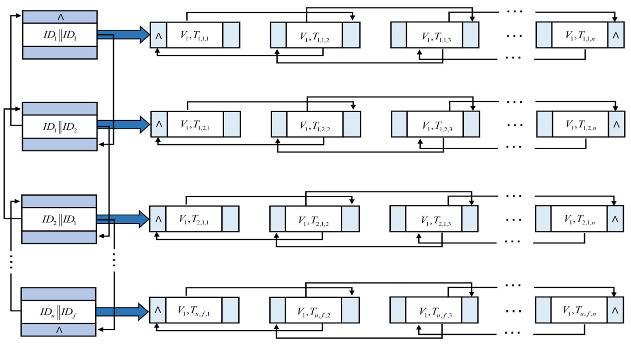


FIGURE 2. Doubly Linked Information Table.

DLIT and LA support an efficient data dynamic structure for data auditing scheme. We conduct file operation and block operation from insertion, deletion, and modification by using the DLIT and LA, which can ensure that the operated data block can still pass the original auditing scheme, while the old data block cannot pass the verification, on other words, only the latest version of the data can pass the verification.

D. VERIFICATION ALGORITHM

1) GLOBAL VERIFICATION ALGORITHM

As mentioned in [25], "global" means all data blocks, "global verification" means that all data blocks should be verified to ensure the integrity of the data. when generating proofs for all data blocks, the workload of global verification is huge for CSP. Since TPA and DO are on the same side, DO may collude with TPA, that is, malicious DO may upload wrong data to prevent CSP from passing verification to obtain CSP's compensation and damage CSP's reputation. So, CSP always uses global verification at the moment that the data file is uploaded or updated to the cloud.

In our construction, we use global verification in the auditing phase, when the CSP received the data file uploaded by the DO, the CSP would use global verification to ensure the integrity of the data file. If the data file can pass the verification, CSP will submit the *Contract* T_1 to the smart contract platform to confirm that the data file has been accepted by CSP.

2) TSV-TRADITIONAL SAMPLING VERIFICATION ALGORITHM

Traditional sampling verification adopts repetitive sampling. In each auditing task, TPA randomly selects some data blocks to check, let the quality of the sample represent the quality of the whole, which greatly reduces the burden of CSP compared to the global verification. As proposed in [26], they use traditional sampling verification to probabilistically detect whether the data contains corrupted blocks. Assuming that the data file is divided into n blocks, and there are k corrupted blocks, the auditor should sample c blocks to ask a data possession proof during the challenge. X donates the number of corrupted blocks being sampled, and P_X donates the probability of successful detection of CSP's misbehavior. Therefore we have:

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

For $n = 10$, $k = 2$, $c = 4$, we have $P_X = 0.66666666667$. Since $\frac{n-i-k}{n-i} < \frac{n-i-1-k}{n-i-i}$, it follows:

$$1 - (\frac{n-k}{n})^c \leq P_X \leq 1 - (\frac{n-c+1-k}{n-c+1})^c \qquad (1)$$

So, we can at least detect corrupt blocks with probability $1 - (\frac{n-k}{n})^c$. The core of this verification is randomness, but randomness is difficult to guarantee in the scheme that the challenge set generated by CSP, as described in [14]. Since each consensus node act as a verifier, the CSP should generate a random challenge by itself, which gives CSP greater authority. A malicious CSP can choose the appropriate c by itself to generate the challenge set it wants to cover up the missing data, which is very unfavorable for the DO. Desiring for a new sampling verification to optimize sampling verification.

IV. NON-INTERACTIVE PUBLIC DATA POSSESSION SCHEME

A. THE SYSTEM MODEL

As shown in Fig. 3, the non-interactive public auditing scheme with fair payment (NI-PAFP) contains three entities: the CSP, the DO, and the verifier.

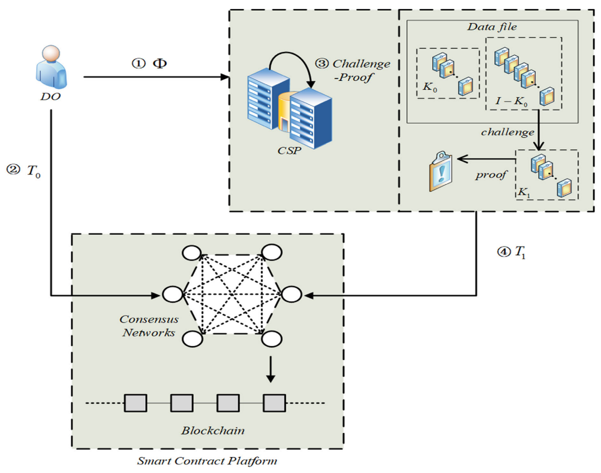


FIGURE 3. The system model.

- ***DO***: the owner of data, who needs the cloud to provide data storage services due to its capabilities and equipment limitations.
- ***CSP***: the storage service and computing service provider for DO.
- ***Verifier***: all the consensus nodes in the smart contract role as a TPA to check the integrity of data files stored in the cloud.

Definition 3 (Non-Interactive Public Auditing Scheme With Fair Payment (NI-PAFP)): A NI-PAFP scheme is composed of five polynomial-time algorithms (***KeyGen, Filepro, ProofGen, Verify, Payment***) such that:

- ***KeyGen*** $(1^{\lambda}) \rightarrow (pk, sk)$: the key generation algorithm takes security parameter λ as input, outputs a key pair (pk, sk), which is run by the DO.
- ***Filepro*** $(U_{ID}, F_{ID}, F, V_i, T_i) \rightarrow$ (Φ, DLIT, LA): the file preprocessing algorithm is run by the DO, which takes the DO's ID U_{ID}, the data file's ID F_{ID}, the data file F, the version of data block V_i, and the time stamp T_i as input, outputs the corresponding data tags Φ, DLIT, and LA.
- ***ProofGen*** $(\tau, c, K_0, F, \Phi) \rightarrow$ *proof*: the proof generation algorithm takes the current state τ, the challenge number c, the set K_0, the data file F, and the data tags Φ as input outputs a data possession proof *proof*, which is run by the CSP. The challenge number is selected in advance by DO, as shown in Table. 2. We suppose τ is the time-varying public information and it cannot be controlled by CSP.
- ***Verify*** $(pk, proof) \rightarrow$ 0 or 1: the verification algorithm takes public key pk and the *proof* as input and outputs 0 or 1, which is run by the verifier and the algorithm can be executed by anyone in the system.
- ***Payment*** (*proof*, *Contract* T_2)$\rightarrow$ *pay*: the payment algorithm takes *proof* and *Contract* T_2 as input, it will activate *Contract* T_0 or *Contract* T_1 based on the result of the ***Verify*** algorithm, which is run by CSP periodically.

TABLE 2. *Contract* T_0.

Contract T_0	
DO ID: U_{ID}	File ID: F_{ID}
File Name: *name*	File Hash: FH
File Size: FS	Upload Time: UT
Storage Period: SP	Service Charges: SC
DO's Account: DOA	CSP's Account: CSA
DO's Public Key: pk_D	DO's Signature: sig_D
Challenged Number: c	
Contract content:	
✓ promise:	
$\diamondsuit\ If(NI-PAFP.Verify(pk_D, proof)) == 1$	
$\rightsquigarrow$ pay SC from DOA to CSA;	

A NI-PDPFP system can be constructed into three phases,

▷ ***Setup phase***
The DO runs ***KeyGen*** and ***Filepro*** algorithms for all data blocks, and sends $\{pk, F, \Phi\}$ to CSP, with keeping sk secret. Meanwhile, DO maintains DLIT and LA in the blockchain and submits *Contract* T_0 to the smart contract platform.

▷ ***Audit phase***
This phase is a non-interactive challenge-proof phase, occurring between CSP and verifier. Upon receiving Φ, CSP will check the integrity of data, and the authentication of the source. If all the tests are qualified, CSP will submit the *Contract* T_1 to the smart contract platform. After that, CSP runs the ***ProofGen*** algorithm to generate a data possession *proof* for the data stored on the cloud, and sends the *proof* to the verifier, then verifier runs the ***Verify*** algorithm to check the validity of the *proof*.

▷ ***Payment phase***
In this phase, the CSP periodically runs ***Payment*** algorithm to get service charges.

As mentioned above, the smart contract T_0 is initiated by the DO and is mainly used for honest CSP to obtain storage service fees from the DO shown in Table. 2 (where the size of the challenge set c is chosen by the user, different challenge set sizes will lead to different detection probabilities and different computational costs and the corresponding service fees will also be different). The smart contract T_1 is initiated by the CSP and is mainly used to punish the malicious behavior of the CSP and hand the fine to the DO shown in Table. 3. The smart contract T_2 is used to activate one of contract T_0 or T_1 to take effect, and the verification algorithm is used to select the activation contract to verify the correctness of the data proof provided by the CSP. If the verification is passed, the contract T_0 will be activated, indicating that the CSP has correctly stored the DO's data and can obtain the corresponding service fee. Otherwise, if the verification fails, the contract T_1 will be activated, impacting that the CSP did not store the data correctly and the data was lost. CSP must compensate DO shown in Table. 4.

B. THREAT MODEL

Generally, the threat in our scheme comes from the compromised CSP and curious verifier.

TABLE 3. *Contract T_1.*

Contract T_1	
File ID: F_{ID}	File Name: *name*
File Hash: FH	File Size: FS
Receiving Time: UT	Storage Period: SP
Penalty: Pen	DO's Account: DOA
CSP's Account: CSA	CSP's Public Key: pk_C
CSP's Signature: sig_C	Challenged Number: c
Contract content: ✓ confirm F; /*T_0 takes effect*/ ✓ promise: $\diamondsuit\ If(NI-PAFP.Verify(pk_D, proof)) == 0$ $\rightsquigarrow$ pay Pen from CSA to DOA;	

TABLE 4. *Contract T_2.*

Contract T_2
Payment contract T_0 Compensation contract T_1 Data possession proof: *proof*
Contract content: $\rightsquigarrow$ activate T_0 or T_1 depended on the result of validating the *proof*;

For the malicious CSP, it may subjectively delete some data rarely accessed for DO to save storage space, or it may cause data loss due to some objective problems such as hardware failure, management errors, or internal and external attacks. To maintain the reputation of the cloud, or unwilling to compensate users, regardless of the motivation, the CSP will not honestly tell users the incident of data loss. In this case, the integrity of the data cannot be guaranteed, and the scheme may face attacks from the malicious CSP as the forgery attack, the replay attack, and the replacing attack.

- ***Forgery attack***: the CSP may forge authentications for corrupted data blocks to deceive the verifier.
- ***Replay attack***: the CSP responds to the challenge with the previous *proof* to pass the verification.
- ***Replacing attack***: the CSP attempts to replace the damaged data block and the corresponding tag with another correct one to pass the verification.

For a curious verifier, it may use a knowledge extractor to obtain data-related information from the data possession proof provided by the CSP. To protect the privacy of data, the scheme needs to achieve privacy-preserving.

C. SECURITY OBJECTIVES

- ***Correctness***: to ensure that if the CSP stores the data file correctly, the *proof* it provides can pass the verification to obtain payment.
- ***Soundness***: to ensure that if the CSP does not store the data properly, it cannot pass the verification and have to compensate DO for the data loss.
- ***Privacy preserved***: to assure that the verifier cannot obtain any valid information about the data during auditing.
- ***Non-interactive***: to ensure that in the auditing phase the CSP doesn't need to interact with the verifier.
- ***Public auditing***: everyone can check the integrity of the data with DO's public key by verifying the *proof* provided by the CSP.
- ***Storage freshness***: to ensure that DO always keeps the latest version of data stored on the cloud.
- ***Dynamic operations support***: to support that DO can update (including insert, delete, modify) the data uploaded to the cloud, which ensures that after updating the data can pass public auditing.
- ***Batch auditing***: to allow that the verifier can complete auditing tasks from multiple users at the same time.

D. SECURITY MODEL

Our security model using a *Game* between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ to define the formal definition of data integrity. Intuitively, the *Game* indicates that $\mathcal{A}$ cannot successfully generate a valid proof without storing all the data blocks properly corresponding to a given challenge unless it guesses all the missing data blocks (as shown in Table. 5).

TABLE 5. *Game Adversary-Forge.*

Game. • $\mathcal{A} \xleftarrow{pk} \textbf{\textit{KeyGen}}_{\mathcal{C}}(1^{\lambda})$
Repeat • $\mathcal{A} \xleftarrow{\Phi} \textbf{\textit{Filepro}}_{\mathcal{C}}(U_{ID}, F_{ID}, F, V_i, T_i)$
Forge • $\mathcal{C} \xleftarrow{proof} \textbf{\textit{ProofGen}}_{\mathcal{A}}(\tau, c, K_0, F, \Phi)$
Return ***Verify*** $(pk, proof)$

- **Setup**: $\mathcal{C}$ runs ***KeyGen*** algorithm, and sends pk to $\mathcal{A}$.
- **Query**: $\mathcal{A}$ makes queries to $\mathcal{C}$ for some files adaptively, $\mathcal{C}$ runs the ***Filepro*** algorithm for these files and returns Φ to $\mathcal{A}$.
- **Forge**: $\mathcal{A}$ outputs *proof* for the data file F with the data tag Φ on state τ, $\mathcal{C}$ returns the result of ***Verify*** algorithm.

If ***Verify*** algorithm outputs 1 with non-negligible probability, donates that $Adv_{\mathcal{A}} = Pr[\textbf{\textit{Verify}}(pk, proof) = 1]$ is non-negligible, then we say that $\mathcal{A}$ wins the above game.

Definition 4: A NI-PDPFP scheme is soundness if any probabilistic polynomial time adversary $\mathcal{A}$ wins the above game for the data file F with data tag Φ on state τ with a non-negligible probability, then the challenger $\mathcal{C}$ can extract the data file by using a knowledge extractor $\textbf{\textit{Extr}}(pk, \sigma, \mu') \rightarrow F$.

V. THE PROPOSED EFFICIENT SAMPLING VERIFICATION ALGORITHM-ESV

The randomness of random sampling like traditional sampling verification makes CSP unable to predict which data blocks will be detected, which greatly improves the detection rate of malicious CSP. But on the other hand, since every detection is random, so a data block may be detected repeatedly or not detected once. A malicious CSP may delete some data blocks that are not frequently detected. In this case, it may not be detected for a long time. Once the timeliness is lost, DO cannot recover it, which will cause permanent loss of data for DO. To reduce the risk of this randomness, we propose an Efficient Sampling Verification.

The core idea of our sampling verification algorithm is to perform random challenges in data blocks outside the previous challenge set to achieve the purpose of narrowing the scope of auditing. Even if a new corrupted block appears during the first auditing, after the second auditing, all corrupted blocks will be gathered in the next data set to be sampled, which means that the third auditing challenge set will be chosen in the data set that contains the first challenge set but not includes the second challenge set.

Assumptions are same as Section III-D2, we donate the first challenge set is $K_0 = \{s'_1, s'_2, \cdots, s'_c\}$ and the probability of the first auditing successful detection of CSP's misbehavior is p_1, the second auditing challenge set is $K_1 = \{s_1, s_2, \cdots, s_c\}$ and the probability is p_2, the probability of the third auditing successful detection is p_3, since the first auditing challenge uses traditional sampling verification, so, $p_1 = P_X$. If no corrupted block is detected in the first auditing, that is, the first challenged blocks are all qualified. When the second auditing starts, the challenge set will randomly choose from $I - K_0$, at this time, the sampling population reduces from n to $n - c$ data blocks, with (1), we can get

$$P_X \le p_2 = 1 - \frac{n-c-k}{n-c} \times \cdots \times \frac{n-2c+1-k}{n-2c+1}$$

So, we can at least detect corrupt blocks with probability $1 - (\frac{n-c+1-k}{n-c+1})^c$. Taking $n = 10, k = 2, c = 4$ as an example, the occurrence of many corrupted blocks can be decomposed into a single one, so here we only describe the occurrence of a single corrupted block. We must consider the following three situations (shown in Fig. 4):

1) No new corrupted block occurs after the first auditing. Using the above equations, we can compute as:

$$\begin{aligned} p_1 &= P_X \\ &= 1 - \frac{n-k}{n} \times \cdots \times \frac{n-c+1-k}{n-c+1} \\ &= 0.66666666667 \\ p_2 &= p_3 \\ &= 1 - \frac{n-c-k}{n-c} \times \cdots \times \frac{n-2c+1-k}{n-2c+1} \\ &= 0.93333333333 \end{aligned}$$

We have $p_3 = p_2 > P_X = p_1$.

2) A corrupted data block arises at $I - K_0$ after the first auditing, then we have:

$$\begin{aligned} P_X &= 1 - \frac{n-k-1}{n} \times \cdots \times \frac{n-c+1-k-1}{n-c+1} \\ &= 0.83333333333 \\ p_1 &= 1 - \frac{n-k}{n} \times \cdots \times \frac{n-c+1-k}{n-c+1} \\ &= 0.66666666667 \end{aligned}$$

during the first auditing, if no corrupted blocks are detected, it means that the first challenged 4 data blocks are all qualified. Therefore, the 2 corrupted blocks are concentrated in the 6 data blocks of $I - K_0$. In the second inspection, a corrupted block arises in $I - K_0$, so there are currently 6 blocks in $I - K_0$, of which 3 are corrupted blocks and 4 data blocks need to be extracted for detection, so the probability is 1, that is, at least one data block will be the corrupted one, so we have $p_2 = p_3 = 1$. So, we can conclude that $p_3 = p_2 > P_X$.

3) A corrupted block occurs at K_0 after the first auditing. So before the second auditing, there are 2 corrupt blocks in $I - K_0$ and 1 corrupt block in K_0. The second auditing is equivalent to extracting 4 data blocks from $I - K_0$ containing 2 corrupt blocks if the corrupt blocks are still not detected. Then the third auditing is equivalent to extracting 4 data blocks from the remaining 6 data blocks containing 3 corrupt blocks, at least one corrupt block will be selected for the third auditing, so the probability of successful detection is 1.

$$\begin{aligned} p_1 &= 0.66666666667 \\ p_2 &= 1 - \frac{n-c-k}{n-c} \times \cdots \times \frac{n-2c+1-k}{n-2c+1} \\ &= 0.93333333333 \\ p_3 &= 1 \end{aligned}$$

$P_X = 0.83333333333$, so we have $p_3 > p_2 = P_X$.

In general, our sampling verification can always make all the corrupted blocks, including the newly appeared ones, gathered in the third sample population. Essentially, the number of corrupted blocks are increased on the premise of reducing the total number of samples, which greatly improves the probability of successful detection compared to the traditional sampling verification algorithm.

VI. THE PROPOSED AUDITING PROTOCOL

In this section, our public auditing scheme is constructed, based on [14]'s NI-PPDP scheme. Our scheme is divided into three phases: the setup phase, the audit phase, and the payment phase.

A. OUR CONSTRUCTION

The details of our construction are as follows:

- ***Setup phase:***

  KeyGen$(1^\lambda) \rightarrow$ (*pk,sk*): DO sets $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, selects g, u as generators of group $\mathbb{G}$, chooses hash functions $H_1(\cdot) : \{0,1\}^* \rightarrow \mathbb{G}$, $H_2(\cdot) : \mathbb{G}_T \rightarrow \mathbb{Z}_p$, $h(\cdot) : \mathbb{Z}_p \rightarrow \mathbb{Z}_p$, randomly generates a signing key pair (ssk, spk), and selects a pseudorandom function $\mathcal{F}(\cdot) : \{I - K_0\} \rightarrow \{I - K_0\}$. Note that the function of $\mathcal{F}(\cdot)$ is randomly permutates the elements in $\{I - K_0\}$. Then, it chooses $x \in \mathbb{Z}_p$, computes as $v = g^x$. Consequently, the DO will hold the secret key $sk = (x, ssk)$ and announce the system public key $pk = (spk, g, u, v, e, H_1(\cdot), H_2(\cdot), \mathcal{F}(\cdot))$.

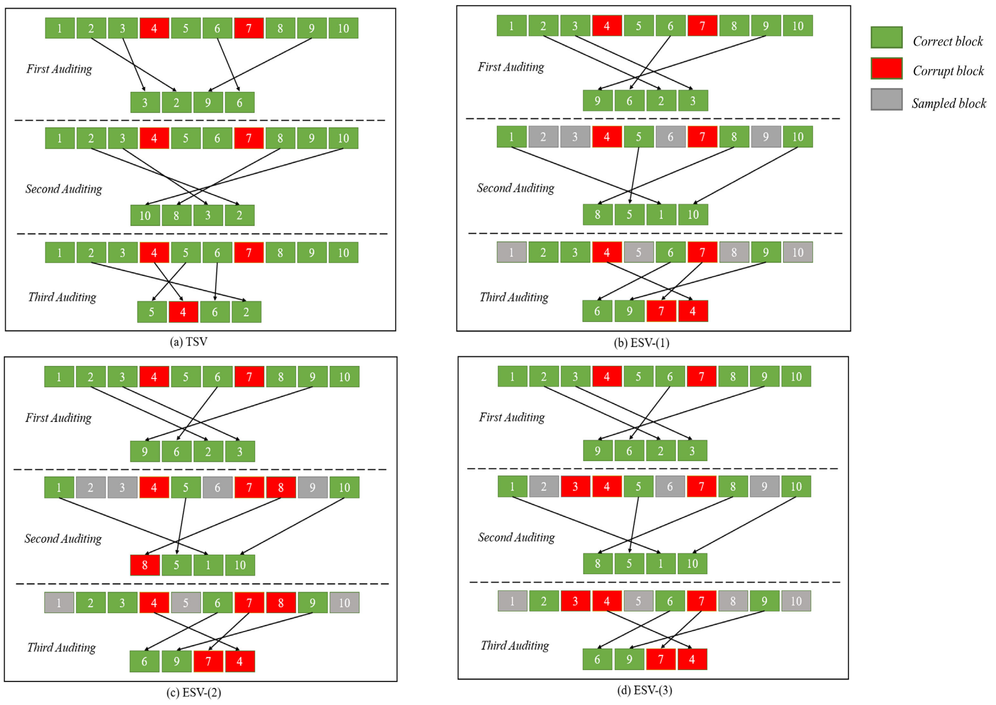


FIGURE 4. Sampling process of TSV and ESV.

Filepro$(U_{ID}, F_{ID}, F, V_i, T_i) \rightarrow (\Phi, DLIT, LA)$: first, DO divides the data file into n blocks as $F = \{m_1, m_2, \cdots, m_n\}$ (where $m_i \in \mathbb{Z}_p$). Second, it computes authenticators for all blocks as $\sigma_i = (H_1(V_i \| T_i) \cdot u^{m_i})^x$, where V_i is m_i's version number, T_i is the current time stamp. Third, DO sets $\varpi = U_{ID} \| F_{ID} \| SIG(U_{ID} \| F_{ID})_{ssk}$ as the file tags. Finally, DO uploads $\Phi = \{F, \{\sigma_i\}, \varpi\}$ to the cloud for storage.

DO maintains the DLIT and the LA by respectively recording the F_{ID}, U_{ID}, V_i, T_i to DLIT, and Loc_i to LA, then it uploads the DLIT and LA to the blockchain. At the same time, the DO submits the *Contract* T_0 to the smart contract platform. Our *Contract* T_0 is only a little different with *Contract* T_0 in [14] that is our *Contract* T_0 contains the challenged number c chosen by DO to delegate the auditing task.

- ***Audit phase:***

ProofGen$(\tau, c, K_0, F, \Phi) \rightarrow proof$: CSP uses τ as the seed of the input secret key of pseudo-random function $\mathcal{F}(\cdot)$. For $j \in \{I - K_0\}$, CSP computes $s_j = \mathcal{F}_\tau(j)$, $K_1 = \{s_1, s_2, \cdots, s_c\}$ is a c-element subset in $\{I - K_0\}$. For $i \in K_1$, CSP calculates $v_i = h(\tau \| i)$, and uses $chal = \{i, v_i\}_{i \in K_1}$ as challenge set to generate the corresponding *proof*. CSP calculates

$$\sigma = \prod_{i \in K_1} \sigma_i^{v_i}, \ \mu' = \sum_{i \in K_1} v_i \cdot m_i. \tag{2}$$

and randomly chooses $s \in \mathbb{Z}_p$, followed by computing $T = e(u, v)^s \in \mathbb{G}_T$, $\mu = s + \gamma\mu' \ mod \ p$, where $\gamma = H_2(T) \in \mathbb{Z}_p$. After that CSP responses to the verifier with $proof = \{\sigma, \mu, T, \tau\}$.

Verify$(pk, proof) \rightarrow 0$ or 1: first, the verifier extracts the ϖ from Φ to verify the integrity of the file name and verifies the authentication of state information τ via blockchain. If any verifications fail, it will abort. Otherwise, the verifier recovers the *name* and computes as follows:

$$K_1 = \{\mathcal{F}_\tau(i) \mid i \in [1, c]\}, \ v_i = h(\tau \| i), \gamma = H_2(T). \tag{3}$$

According to the data information stored in the DLIT, the verifier calculates

$$D_i = H_1(V_i \| T_i), \quad D = \prod_{i \in K_1} D_i^{v_i} \tag{4}$$

Finally, the verifier checks whether

$$T \cdot e(\sigma^\gamma, g) = e(D^\gamma, v) \cdot e(u^\mu, v) \tag{5}$$

If it holds, K_1 will be recorded in blockchain as K_0 for next auditing and output 1, where assuming the initial value of K_0 is $\{0, \cdots 0\}$, else it will output 0.

- ***Payment phase***:

Payment $(proof, \textit{Contract } T_2) \rightarrow pay$: CSP submits the *Contract* T_2, which contains a *proof*. If validation is successful, the *Contract* T_0 will be activated and DO's service fees will be automatically paid to CSP, else the *Contract* T_1 will be activated and CSP's compensation will be paid to DO.

B. CORRECTNESS

$$\begin{aligned}T \cdot e(\sigma^{\gamma}, g) &= e(u, v)^{s} \cdot e((\prod_{i \in K_1}(H_1(V_i \| T_i) \cdot u^{m_i})^{x \cdot v_i})^{\gamma}, g) \\ &= e((\prod_{i \in K_1} D_i^{v_i} \cdot u^{m_i \cdot v_i})^{\gamma} \cdot u^{s}, v) \\ &= e(\prod_{i \in K_1} D_i^{\gamma \cdot v_i} \cdot u^{m_i \cdot v_i \cdot \gamma + s}), v) \\ &= e(\prod_{i \in K_1} D_i^{v_i \cdot \gamma}, v) \cdot e(\prod_{i \in K_1} u^{m_i v_i \gamma + s}, v) \\ &= e(D^{\gamma}, v) \cdot e(u^{\mu}, v)\end{aligned}$$

C. DYNAMIC AUDITING

To realize the function of dynamic auditing for the data [21], our scheme introduces DLIT and LA to perform dynamic operations such as inserting, modifying, and deleting. However, unlike scheme [25], we adopt a non-interactive way and uses consensus nodes to replace TPA. To reduce communication overhead, the DO should maintain DLIT and LA in the blockchain. The dynamic operation (where *operate* is selected from *insert*, *delete*, and *modify*) workflow is as follows: (as shown in Fig. 5)

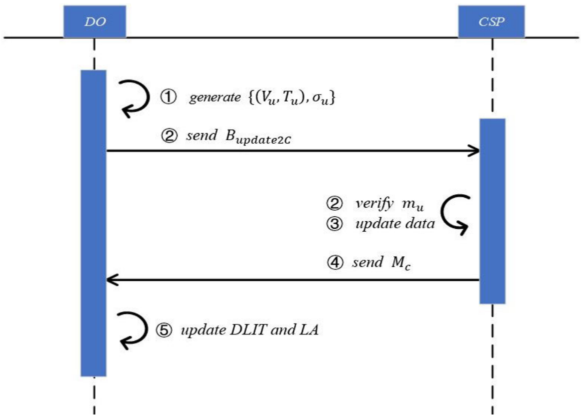


FIGURE 5. Dynamic auditing.

1) DO generates a version V_u, a time stamp T_u, and a signature σ_u for the data block m_u needed to be updated.
2) DO sends $B_{update2C} = \{operate, U_I D, F_I D, i, m_u, \sigma_u\}$ to the CSP.
3) CSP verifies m_u with σ_u when receiving the DO's operation requisition $B_{update2C}$.
4) If the verification is successful, CSP will update the data and send $M_c = \{True\}$ to DO.
5) Only when DO receives M_c, can it update the DLIT and LA.

Take the data block insert operation as an example. Assuming that the user ID_1 wants to add a data block m_x after the data block m_1 in the file ID_2, the DLIT and LA will perform the following update operations.

1) Updating DLIT. First, disconnecting the two links between the data information of m_1 and m_2. Secondly, inserting the data information $V_x, T_{1,2,x}$ corresponding to m_x between the data information of m_1 and m_2, where m_x is a new data block, so $V_x = 1$. Finally, linking the data information of m_x with the data information of m_1 and the data information of m_2 respectively (as shown in Fig. 6).

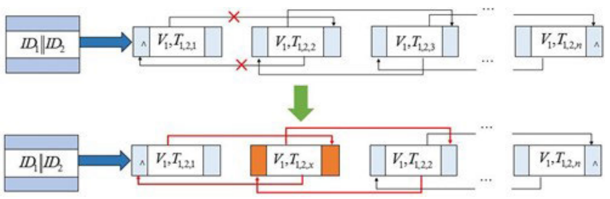


FIGURE 6. Insert operation in DLIT.

2) Updating LA. Inserting m_x directly between m_1 and m_2. After adding the data block m_x, the content corresponding to position $b_{2,1}$ is m_1, the content of position $b_{2,2}$ is m_x, and the content of position $b_{2,3}$ is m_2 (as shown in Fig. 7). That is, m_x is the second data block of the second file, which corresponds to the positional relationship in DLIT.

$$\begin{matrix}(m_1 & m_2 & m_3 & \cdots & m_n) \\ \uparrow & \uparrow & \uparrow & & \uparrow \\ b_{2,1} & b_{2,2} & b_{2,3} & \cdots & b_{2,n}\end{matrix} \Rightarrow \begin{matrix}(m_1 & m_x & m_2 & m_3 & \cdots & m_n) \\ \uparrow & \uparrow & \uparrow & \uparrow & & \uparrow \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & \cdots & b_{2,n}\end{matrix}$$

FIGURE 7. Insert operation in LA.

D. BATCH AUDITING

In the blockchain, there are many auditing tasks for a verifier. Batch auditing can work efficiently in those scenarios that multi-user storage and multi-file stored by one user. For the multi-user storage scenario, suppose there are users $u_i \in \{u_1, \cdots, u_s\} \triangleq \mathcal{U}$ with the corresponding stored file $f_i \in \{f_1, \cdots, f_s\} \triangleq \Omega$, then the verifier only needs to verify whether the following equation holds,

$$\prod_{u_i \in \mathcal{U}} \prod_{f_i \in \Omega} T \cdot e(\sigma^{\gamma}, g) = \prod_{u_i \in \mathcal{U}} \prod_{f_i \in \Omega} e(D^{\gamma}, v) \cdot e(u^{\mu}, v)$$

If it holds, all the verified files are stored completely and correctly on the cloud. But if the following equation does not hold, the batch auditing is invalid, the verifier needs to audit one by one to find the corrupted files. In the multi-file stored by one user scenario, the verifier needs to verify:

$$\prod_{f_i \in \Omega} T \cdot e(\sigma^{\gamma}, g) = \prod_{f_i \in \Omega} e(D^{\gamma}, v) \cdot e(u^{\mu}, v)$$

If it holds, the integrity of all the verified files for one user can be ensured. otherwise, the verifier needs to re-verify all files to find out the corrupted one.

VII. SECURITY ANALYSIS

Theorem 1 (Soundness): No probabilistic polynomial-time adversary can forge a data possession proof to pass the verification with non-negligible probability if the computational Diffie-Hellman assumption holds.

Proof: As proved in [27], We prove the soundness of our scheme by using a sequence of games between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$, as defined follows:

Game 0. *Game 0* is defined in Table.5.

Game 1. *Game 1* is the same as *Game 0*, except that all the tags signed by $\mathcal{C}$ will be recorded in a list, if $\mathcal{A}$ sends a tag, which is a valid signature under the *ssk* but it is not signed by $\mathcal{C}$, then $\mathcal{C}$ will abort the game.

Game 2. *Game 2* is same as *Game 1*, only one difference between them, that is, in *Game 2*, $\mathcal{C}$ keeps a list of the responses to $\mathcal{A}$'s inquiries, if $\mathcal{A}$ passes the verification with a forgery $\sigma*$ ($\sigma* \neq \sigma$, σ is the correct aggregate signature), $\mathcal{C}$ will abort the game.

Game 3. *Game 3* is the same as *Game 2*, excepting that if $\mathcal{A}$ submits a $\mu*$, which is not equal to the expected μ, $\mathcal{C}$ will abort the game.

Lemma 1: If there exists an algorithm B can distinguish *Game 0* and *Game 1* with non-negligible probability, then we can build an algorithm C against the unforgeability of BLS-Homomorphic Verifiable Authenticator's (BLS-HVA's) existence with non-negligible advantage.

Proof: As shown in [8] and [28], based on the hardness of CDH assumption, the BLS-HVA cannot be forged by any adversary.

Lemma 2: If there exists an algorithm B can distinguish *Game 1* and *Game 2* with non-negligible probability, then we can construct an algorithm C against the CDH assumption with non-negligible advantage.

Proof: In *Game 2*, assuming that what $\mathcal{C}$ expected is $\{\sigma, \mu\}$ with $\mu = s + \gamma\mu'$, while $\mathcal{A}$ submits is $\{\sigma^*, \mu^*\}$ where $\mu^* = s + \gamma\mu'^*$. Obviously, $\mu'^* \neq \mu'$, otherwise $\sigma^* = \sigma$, defines $\triangle\mu = \mu - \mu^*$.

With this adversary, the simulator could break the CDH instance as follows:

Given $(g, g^x, h) \in G$, the simulator needs to output h^x.

The simulator sets $v = g^x$, $u = g^a h^b$. If $\mathcal{A}$ can pass the verification, we can construct as follows:

$$\begin{cases} T \cdot e(\sigma^\gamma, g) = e(D^\gamma, v) \cdot e(u^\mu, v) \\ T \cdot e(\sigma^{*\gamma}, g) = e(D^\gamma, v) \cdot e(u^{\mu^*}, v) \end{cases} \tag{6}$$

We can obtain that

$$\begin{aligned} e((\sigma/\sigma^*)^\gamma, g) &= e(u^{\triangle\mu}, v) = e((g^a h^b)^{x\triangle\mu}, g) \\ e(h^{b\triangle\mu}, g^x) &= e((\sigma/\sigma^*)^\gamma \cdot v^{-a\triangle\mu}, g) \\ e(h^x, g) &= e((\sigma/\sigma^*)^\gamma \cdot v^{-\frac{a\triangle\mu}{b\triangle\mu}}, g) \\ h^x &= (\sigma/\sigma^*)^\gamma \cdot v^{-\frac{a\triangle\mu}{b\triangle\mu}} \end{aligned}$$

where the probability that $b\triangle\mu = 0 \; mod \; p$ is only $1/p$, which is negligible.

Lemma 3: If there exists an algorithm B can distinguish *Game 2* and *Game 3* with non-negligible probability, then we can create an algorithm C against the DL assumption with non-negligible advantage.

Proof: As [7] proved, in *Game 3*, assuming that what $\mathcal{C}$ expected is $\{\sigma, \mu\}$ with $\mu = s + \gamma\mu'$, while $\mathcal{A}$ submits is $\{\sigma^*, \mu^*\}$ where $\mu^* = s + \gamma^*\mu'^*$. As demonstrated in *Game 2*: $\sigma = \sigma^*$, so we can conclude that $\mu' = \mu'^*$. As we analyze above, $\mathcal{A}$ submits a forged proof $\{\sigma, \mu^*\}$, where $\mu^* = s + \gamma^*\mu'$, defines $\triangle\mu = \mu - \mu^*$. If $\mathcal{A}$ can pass the verification, we can construct:

$$\begin{cases} T \cdot e(\sigma^\gamma, g) = e(D^\gamma, v) \cdot e(u^\mu, v) \\ T \cdot e(\sigma^{\gamma^*}, g) = e(D^{\gamma^*}, v) \cdot e(u^{\mu^*}, v) \end{cases} \tag{7}$$

we can calculate as follows:

$$\begin{aligned} e(\sigma^{\gamma-\gamma^*}, g) &= e(D^{\gamma-\gamma^*} \cdot u^{\triangle\mu}, v) \\ e((\prod_{i\in K_1} D_i \cdot u^{m_i})^{v_i(\gamma-\gamma^*)}, v) &= e(\prod_{i\in K_1} D_i^{v_i(\gamma-\gamma^*)} \cdot u^{\triangle\mu}, v) \\ e(u^{\sum_{i\in K_1} m_i v_i(\gamma-\gamma^*)}, v) &= e(u^{\triangle\mu}, v) \\ u^{\frac{(\gamma-\gamma^*)}{\triangle\mu}\sum_{i\in K_1} m_i v_i} &= 1 \end{aligned}$$

In this case, the probability of $\triangle\mu = 0 \; mod \; p$ is negligible as we mentioned above. What's more, we can compute that

$$\sum_{i\in K_1} m_i v_i = \frac{\triangle\mu}{\gamma-\gamma^*} = \frac{\mu-\mu^*}{\gamma-\gamma^*}$$

If $\mathcal{A}$ can pass the verification, we can response $\{pk, \sigma, \frac{\mu-\mu^*}{\gamma-\gamma^*}\}$ as input for ***Extr***(pk, σ, μ') to recover the data file F. As the game sequence and Lemma 1-Lemma 3 analyzed above, there is only negligible difference probability between these games, which completes the proof.

Theorem 2 (Privacy Preserved): From the *proof* $= \{\sigma, \mu, T, \tau\}$ responded by CSP, the verifier cannot recover μ'. So, our scheme is privacy preserved.

Proof: As shown in [29] and [30], We can construct a simulator that can generate a valid proof for simulation without knowing μ'. The verifier acts as an adversary, who wants to obtain valid information through the proof from the simulator.

The simulator randomly chooses $\gamma, \mu \in \mathbb{Z}_p$, Since the simulator controls the random oracle $h(\cdot)$, then it sets as follows:

$$T = e(D^{\gamma^*}, v) \cdot e(u^{\mu^*}, v)/e(\sigma^{\gamma^*}, g), \quad \gamma = h(T)$$

it can construct a valid proof to the verifier.

VIII. EVALUATION

1) ***Theoretical analysis***

In the process of providing data storage services, the malicious CSP may hide the data loss to protect its reputation by forgery Attacks, replay attacks, replacement attacks, such as [12] and [31] referred in their threat model. Therefore, a great data auditing protocol should be able to successfully detect corrupted

TABLE 6. The computational overhead comparison with existing related schemes.

Scheme	Signature Generation	Proof Generation	Proof Verification
[5]	$2nMul + 2nExp$	$3cMul + 2cExp$	$3cMul + (c+1)Exp + 3Pair$
[9]	$2nMul + 2nExp$	$2cMul + 2cExp$	$(c+l+2)Mul + (c+l+4)Exp + 4Pair$
[14]	$nMul + 2nExp$	$2cMul + cExp$	$(c+1)Mul + (c+3)Exp + 2Pair$
[15]	$nMul + 2nExp$	$2cMul + cExp$	$(c+1)Mul + (3c+2)Exp + 2Pair$
[29]	$nMul + 2nExp$	$2cMul + cExp$	$(c+1)Mul + (c+3)Exp + 3Pair$
Ours	$nMul + 2nExp$	$2cMul + cExp$	$(c+1)Mul + (c+3)Exp + 3Pair$

Note: "n" is the whole number of data blocks in a file; "c" is the number of the challenged blocks when auditing a file; the user's identity ID is "l" bit; "$Pair$" means "pairing operation"; "Mul" means "multiplication operation"; "Exp" means "exponential operation".

blocks under these attacks. As referred to Theorem 1, our scheme can resist the forgery attacks, since the BLS-HVA is computationally unforgeable based on the hardness of the computational Diffie-Hellman problem. Our scheme uses DLIT to record data information and uses the version number V_i and time stamp T_i to generate a signature for each data, which can prevent CSP from adopting previous information or replacing the information to pass the detection. Assuming that the data need to be detected is m_j with V_j, T_i, CSP substitutes m_j, σ_j with m_j*, σ_j*, V_j*, T_j* and generates a fake proof *proof* $= \{\sigma*, \mu*, T, \tau\}$ for detection, where

$$\sigma^* = \prod_{i\in[i,j-1]\cup[j+1,s]} \sigma_i^{v_i} \cdot \sigma_j^{*v_j^*} \tag{8}$$

$$\mu'^* = \sum_{i\in[i,j-1]\cup[j+1,s]} m_i \cdot v_i + m_j^* \cdot v_j^* \tag{9}$$

When the verifier verifies *proof*, it checks whether (5) is held. As we computed in section IX, the left part contains $H_1(V_j^* \parallel T_j^*)$ calculated by the CSP in σ^*, the right part contains $H_1(V_j \parallel T_j)$ computed by the verifier, we can believe that only when (10) holds, can the equation holds, which means (11) holds.

$$H_1(V_j^* \parallel T_j^*) = H_1(V_j \parallel T_j) \tag{10}$$

$$\begin{cases} V_j^* = V_j \\ T_j^* = T_j \end{cases} \tag{11}$$

So, replay attacks and replacement attacks can never succeed under our scheme.

2) ***Computational overhead***

To further evaluate the efficiency of our scheme, we use Java language to write the codes in MyEclipse to simulate our experiment, where the final results are obtained by running the program 50 times and taking the average value, the results obtained are presented graphically using MATLAB R2018. In detail, we adopt the JPBC library and select Type-A elliptic curve with the order of 160-bit to realize the pairing operation. The client-side and cloud service provider side are both operate on Windows 10 with a 3.91 GHz Intel Core i3-7100 CPU and 4 GB of memory. We mainly consider the computational cost of generating signatures, generating proofs, and verifying proofs. Among them, we only take pairing operations, multiplication operations, and exponential operations into account, since compared to these operations costs of others are negligible. We denote the pairing operation, multiplication operation, and exponential operation as *Pair*, *Mul*, and *Exp* respectively.

Fig. 8 shows the time consumption for different numbers of data blocks from 0 to 1000 to generate signatures, proofs, and verify proofs. Our scheme assuming that data auditing is performed on n data blocks, and c blocks need to be randomly selected for verification.

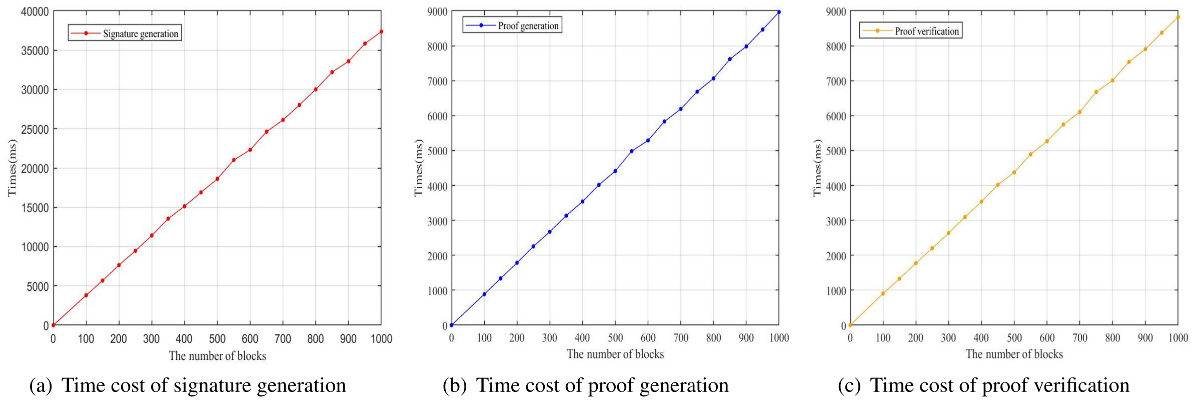
a) In the ***FilePro*** algorithm, DO generates signatures for each data block as $\sigma_i = (H_1(V_i \| T_i) \cdot u^{m_i})^x$. So, the total computational overhead is $nMul + (n+1)Exp$.
b) In the ***ProofGen*** algorithm, CSP generates a proof for c blocks in the challenge set. CSP needs to compute $\sigma = \prod_{i\in K_1} \sigma_i^{v_i}$, $\mu' = \Sigma_{i\in K_1} v_i \cdot m_i$, which costs $cMul + cExp$.
c) After received the *proof* from CSP, the verifier uses the ***Verify*** algorithm to verify the correctness of the proof by checking whether $T \cdot e(\sigma^\gamma, g) = e(D^\gamma, v) \cdot e(u^\mu, v)$ holds, which costs $(c+1)Mul + (c+3)Exp + 2Pair$.

Table. 6 shows the computational overhead comparison between existing related schemes and our scheme. As we expected, since our scheme is still based on the framework proposed by Ateniese *et al.*, both we use the homomorphic verifiable authenticator technique to generate signatures for data, so the difference in the signature generation, proof generation, and proof verification are not obvious.

TABLE 7. Comparison of communication costs and functions.

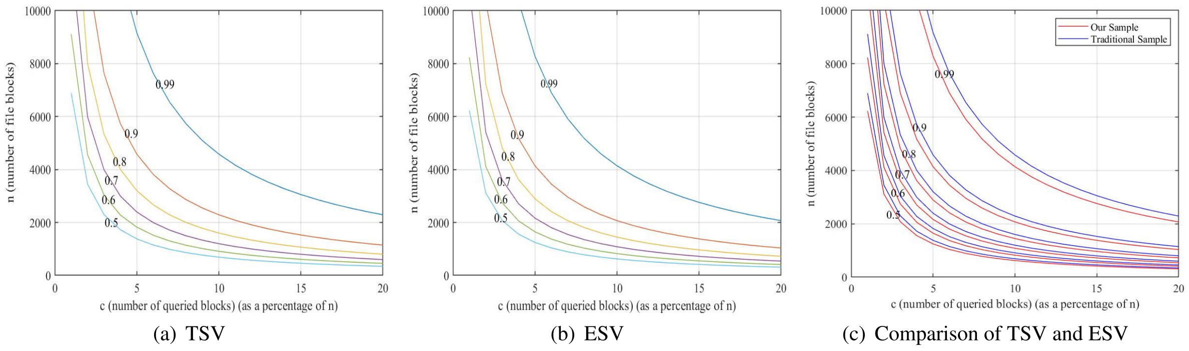
Scheme	Communication Costs	Public Auditing	Dynamic Auditing	Privacy Preserved	Batch Auditing	Detection Probability
DPDP(MHT) [22]	$cO(logn)$	√	√	√	√	$1-(\frac{n-k}{n})^c$
DPDP(skip list) [21]	$cO(logn)$	×	√	×	×	$1-(\frac{n-k}{n})^c$
IHT-PA [23]	$O(c)$	√	√	√	−	$1-(\frac{n-k}{n})^{cs}$
DHT-PA [24]	$O(c)$	√	√	√	√	$1-(\frac{n-k}{n})^c$
DLIT-PA [25]	$O(c)$	√	√	×	√	$1-(\frac{n-k}{n})^c$
Ours	$O(1)$	√	√	√	√	$1-(\frac{n-k-c+1}{n-c+1})^c$

Note: "n" is the whole namber of data blocks in a file; a block is divided into "s" segments; "c" is the number of the challenged blocks when auditing a file; "k" is the number of corrupted blocks. "$\surd$" means "support"; "$\times$" means "not support"; "$-$" means "not mentioned".



(a) Time cost of signature generation (b) Time cost of proof generation (c) Time cost of proof verification

FIGURE 8. Time cost.



(a) TSV (b) ESV (c) Comparison of TSV and ESV

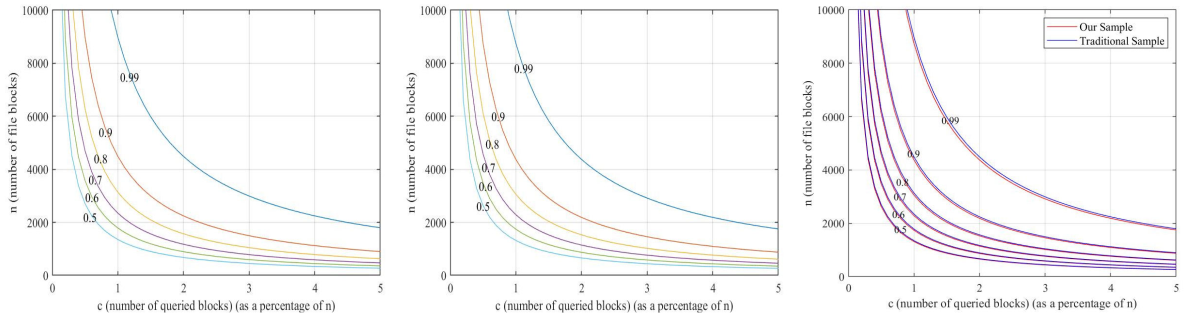
FIGURE 9. k = 1% of n.

3) ***Dynamic auditing schemes comparison of communication costs and functions***

Table. 7 shows the comparison of dynamic auditing schemes in terms of communication overhead and functions. The design of the non-interactive auditing process makes the communication overhead only generated by the CSP to send the proof to the verifier, which costs O(1). And our scheme is the only one, which can achieve all the auditing requirements includes public auditing, dynamic auditing, privacy preserved, batch auditing. The design of the ESV algorithm makes our scheme can avoid The emergence of inefficient audit situations and achieve greater detection probability than other scheme.

4) ***Analyses of Sampling verification algorithm***

Fig. 9 and Fig. 10 describe that when the traditional sampling verification and the proposed sampling verification contain the same number of corrupted blocks, changes in the number of data blocks *n* and the



(a) TSV (b) ESV (c) Comparison of TSV and ESV

FIGURE 10. $k = 5\%$ of n.

number of challenged blocks c (as a percentage of n) under different probabilities.

Assuming that the file blocks contain $k\%$ of n corrupted blocks, the corrupted blocks had not to be detected for the first time, indicating that these corrupted blocks are among the remaining $n - c$ data blocks. When next detection is continued, sampling is performed in the remaining $n - c$ data blocks, which is equivalent to the situation that traditional sampling verification samples c blocks from $n - c$ blocks for detection, so its efficiency will not be lower than the traditional scheme, and the trend should be the same as that of the traditional scheme, as shown in Fig. 9(a), 9(b) and Fig. 10(a), 10(b).

Fig. 9(c), 10(c) depicts the comparison between our efficient sampling verification algorithm and the traditional sampling verification algorithm. It can be seen that the curve graph of ESV is below TSV under different probabilities, that is, under the condition of a certain n, ESV can achieve that by sampling fewer blocks to reach the same probability as TSV. Comparing Fig. 9(c) and Fig. 10(c), it can be found that the gap between the two schemes in Fig. 9(c) that $k = 1\%$ of n is greater than in Fig. 10(c) that $k = 5\%$ of n. In other words, with a smaller proportion of corrupted blocks in the overall number n, the advantages of our ESV will be more obvious to detect corrupted blocks faster. Take extreme cases as an example to illustrate the efficiency of the ESV algorithm. Assuming n=10000, k=1, c=5000, the probability that the TSV algorithm detects the corrupted block in each auditing is 0.3935, while for the ESV algorithm, regardless of whether it is detected in the first auditing, the corrupted block must be detected in the second auditing.

IX. CONCLUSION

In this paper, we design a data auditing scheme with a novel sampling verification algorithm, which ensures that those data blocks sampled at last time will not be sampled for testing during current auditing, and the probability of successful detection is improved. Compared with the traditional sampling verification algorithm, the corrupted block can be detected faster by our sampling verification algorithm. Dynamic auditing function has also been developed for the auditing scheme to help users update data efficiently. The security analysis shows the soundness and security of our scheme, and the performance evaluation shows the efficiency of our scheme.

REFERENCES

[1] (Sep. 2020). *Cloud Storage—Global Market Trajectory and Analytics*. [Online]. Available: https://www.researchandmarkets.com/reports/5140992/cloud-storage-global-market-trajectory-and

[2] (Jan. 17, 2019). *4 Real Life Examples of Data Loss*. [Online]. Available: https://www.strongholddata.com/4-real-life-examples-of-data-loss/

[3] R. Burrows. (Apr. 23, 2013). *The 5 Most Disastrous Data Loss Incidents in Recent History*. [Online]. Available: https://www.business2community.com/big-data/the-5-most-disastrous-data-loss-incidents-in-recent-history-0473611

[4] A. Oprea, M. K. Reiter, and K. Yang, "Space-efficient block storage integrity," in *Proc. NDSS*, 2005, pp. 1–12. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/citations?doi=10.1.1.59.1417

[5] X. Yang, M. Wang, X. Wang, G. Chen, and C. Wang, "Stateless cloud auditing scheme for non-manager dynamic group data with privacy preservation," *IEEE Access*, vol. 8, pp. 212888–212903, 2020, doi: 10.1109/ACCESS.2020.3039981.

[6] Y. Zhang, C. Chen, D. Zheng, R. Guo, and S. Xu, "Shared dynamic data audit supporting anonymous user revocation in cloud storage," *IEEE Access*, vol. 7, pp. 113832–113843, 2019, doi: 10.1109/ACCESS.2019.2935180.

[7] G. Yang, J. Yu, W. Shen, Q. Su, Z. Fu, and R. Hao, "Enabling public auditing for shared data in cloud storage supporting identity privacy and traceability," *J. Syst. Softw.*, vol. 113, pp. 130–139, Mar. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412121500268X

[8] H. Tian, F. Nan, H. Jiang, C.-C. Chang, J. Ning, and Y. Huang, "Public auditing for shared cloud data with efficient and secure group management," *Inf. Sci.*, vol. 472, pp. 107–125, Jan. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025518306996

[9] W. Shen, J. Qin, J. Yu, R. Hao, and J. Hu, "Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 2, pp. 331–346, Feb. 2019, doi: 10.1109/TIFS.2018.2850312.

[10] X. Zhang, H. Wang, and C. Xu, "Identity-based key-exposure resilient cloud storage public auditing scheme from lattices," *Inf. Sci.*, vol. 472, pp. 223–234, Jan. 2019, doi: 10.1016/j.ins.2018.09.013.

[11] R. Ding, Y. Xu, J. Cui, and H. Zhong, "A public auditing protocol for cloud storage system with intrusion-resilience," *IEEE Syst. J.*, vol. 14, no. 1, pp. 633–644, Mar. 2020, doi: 10.1109/JSYST.2019.2923238.

[12] N. Garg and S. Bawa, "Comparative analysis of cloud data integrity auditing protocols," *J. Netw. Comput. Appl.*, vol. 66, pp. 17–32, May 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516300224

[13] D. He, H. Wang, J. Zhang, and L. Wang, "Insecurity of an identity-based public auditing protocol for the outsourced data in cloud storage," *Inf. Sci.*, vol. 375, pp. 48–53, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025516309847

[14] H. Wang, H. Qin, M. Zhao, X. Wei, H. Shen, and W. Susilo, "Blockchain-based fair payment smart contract for public cloud storage auditing," *Inf. Sci.*, vol. 519, pp. 348–362, May 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025520300621

[15] H. Yuan, X. Chen, J. Wang, J. Yuan, H. Yan, and W. Susilo, "Blockchain-based public auditing and secure deduplication with fair arbitration," *Inf. Sci.*, vol. 541, pp. 409–425, Dec. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002002552030668X

[16] Y. Wang, Q. Wu, B. Qin, W. Shi, R. H. Deng, and J. Hu, "Identity-based data outsourcing with comprehensive auditing in clouds," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 4, pp. 940–952, Apr. 2017, doi: 10.1109/TIFS.2016.2646913.

[17] T. Wu, G. Yang, Y. Mu, R. Chen, and S. Xu, "Privacy-enhanced remote data integrity checking with updatable timestamp," *Inf. Sci.*, vol. 527, pp. 210–226, Jul. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002002552030236X

[18] J. Yuan and S. Yu, "Public integrity auditing for dynamic data sharing with multiuser modification," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 8, pp. 1717–1726, Aug. 2015, doi: 10.1109/TIFS.2015.2423264.

[19] Y. Yu, Y. Li, J. Ni, G. Yang, Y. Mu, and W. Susilo, "Comments on 'public integrity auditing for dynamic data sharing with multiuser modification,'" *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 3, pp. 658–659, Mar. 2016, doi: 10.1109/TIFS.2015.2501728.

[20] M. Sookhak, A. Gani, M. K. Khan, and R. Buyya, "WITHDRAWN: Dynamic remote data auditing for securing big data storage in cloud computing," *Inf. Sci.*, vol. 380, pp. 101–116, Feb. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025515006581

[21] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: ACM, 2009, pp. 213–222, doi: 10.1145/1653662.1653688.

[22] 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, doi: 10.1109/TPDS.2010.183.

[23] 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. 2013, doi: 10.1109/TSC.2011.51.

[24] H. Tian, Y. Chen, C.-C. Chang, H. Jiang, Y. Huang, Y. Chen, and J. Liu, "Dynamic-hash-table based public auditing for secure cloud storage," *IEEE Trans. Services Comput.*, vol. 10, no. 5, pp. 701–714, Sep. 2017, doi: 10.1109/TSC.2015.2512589.

[25] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 10, pp. 2402–2415, Oct. 2017, doi: 10.1109/TIFS.2017.2705620.

[26] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: ACM, 2007, pp. 598–609, doi: 10.1145/1315245.1315318.

[27] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Advances in Cryptology— ASIACRYPT 2008*, J. Pieprzyk, Ed. Berlin, Germany: Springer, 2008, pp. 90–107, doi: 10.1007/978-3-540-89255-7_7.

[28] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, Sep. 2004, doi: 10.1007/s00145-004-0314-9.

[29] X. Li, S. Liu, and R. Lu, "Comments on 'A public auditing protocol with novel dynamic structure for cloud data,'" *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2881–2883, 2020, doi: 10.1109/TIFS.2020.2978592.

[30] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 362–375, Feb. 2013, doi: 10.1109/TC.2011.245.

[31] D. Kirovski, F. A. P. Petitcolas, and Z. Landau, "The replacement attack," *IEEE Trans. Audio, Speech Lang. Process.*, vol. 15, no. 6, pp. 1922–1931, Aug. 2007, doi: 10.1109/TASL.2007.900088.

XUELIAN LI received the Ph.D. degree in cryptography, in 2010. She is currently an Associate Professor with the School of Mathematics and Staticstics, Xidian University. Her research interests include information security and blockchain.

LISHA CHEN was born in Xiantao, Hubei, China, in 1996. She is currently pursuing the M.S. degree with Xidian University, Xi'an, China. Her research interests include data trading and cloud storage auditing.

JUNTAO GAO received the Ph.D. degree in cryptography, in 2006. He is currently an Associate Professor with the School of Telecommunication and Engineering, Xidian University. His research interests include pseudorandom sequences and blockchain.