An Efficient Privacy-Preserving Public Auditing Protocol for Cloud-Based Medical Storage System

Xiong Li, *Member, IEEE*, Shanpeng Liu, Rongxing Lu, *Fellow, IEEE*, Muhammad Khurram Khan, *Senior Member, IEEE*, Ke Gu, *Associate Member, IEEE*, and Xiaosong Zhang

***Abstract*—The booming Internet of Things makes smart healthcare a reality, while cloud-based medical storage systems solve the problems of large-scale storage and real-time access of medical data. The integrity of medical data outsourced in cloud-based medical storage systems has become crucial since only complete data can make a correct diagnosis, and public auditing protocol is a key technique to solve this problem. To guarantee the integrity of medical data and reduce the burden of the data owner, we propose an efficient privacy-preserving public auditing protocol for the cloud-based medical storage systems, which supports the functions of batch auditing and dynamic update of data. Detailed security analysis shows that our protocol is secure under the defined security model. In addition, we have conducted extensive performance evaluations, and the results indicate that our protocol not only remarkably reduces the computational costs of both the data owner and the third-party auditor (TPA), but also significantly improves the communication efficiency between the TPA and the cloud server. Specifically, compared with other related work, the computational cost of the TPA in our protocol is negligible and the data owner saves more than $2/3$ of computational cost. In addition, as the number of challenged blocks increases, our protocol saves nearly 90% of communication overhead between the TPA and the cloud server.**

***Index Terms*—Auditing protocol, batch auditing, cloud-based medical storage, integrity, privacy preserving.**

I. Introduction

With the popularity of mobile Internet and the Internet of Things (IoT), the amount of data collected by various devices has grown exponentially, and the era of Big Data has come into our daily lives. According to the IDC's white paper "Data Age 2025" [1], the global data in 2025 is expected to reach 175ZB. The traditional storage method is unable to satisfy the demands for the storage of a huge amount of current data. Luckily, cloud storage [2], [3], which is regarded as a powerful resource system of data storage, can provide users with different pay-as-you-go cloud services remotely without directly being managed by users. Because of its ability to provide users with highly reliable, scalable, and on-demand services at a low price, cloud storage has received widespread attention and has been applied in many real-world scenarios today, and an increasing number of organizations and individuals are migrating their data to cloud storage. Up to date, the booming cloud storage has spawned a series of cloud service providers, such as Amazon AWS, Google Drive, Microsoft Azure, and so on.

The integration of various advanced technologies such as information, telecommunications, and sensors has made smart medical care possible. In the vision of smart healthcare, the medical data of patients is collected by different types of medical sensors and then used by medical staff to diagnose the patients' physical condition. By collecting data from a large number of patients, medical institutions can evaluate the health status of residents in the region and make predictions about current epidemics. As the smart medical system generates a large amount of data, it is a wise choice to outsource the medical data to cloud storage to facilitate users' access to the data and help doctors make real-time diagnosis. Compared with other types of data, medical data has its particularity. On the one hand, medical data contains a large amount of patients' private information. On the other hand, the integrity of medical data is particularly important because it is the basis for making a correct diagnosis.

Manuscript received July 2, 2021; revised October 28, 2021 and December 15, 2021; accepted December 30, 2021. Date of publication January 6, 2022; date of current version May 5, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 62072078, in part by the Sichuan Science and Technology Program under Grant 2020JDTD0007, and by King Saud University, Riyadh, Saudi Arabia under Project number RSP-2021/12. *(Corresponding author: Ke Gu.)*

Xiong Li is with the Institute for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: lixiongzhq@163.com).

Shanpeng Liu is with the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China (e-mail: liushanpeng0@gmail.com).

Rongxing Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton E3B 5A3, Canada (e-mail: rlu1@unb.ca).

Muhammad Khurram Khan is with the College of Computer & Information Sciences, King Saud University, Riyadh 11653, Kingdom of Saudi Arabia (e-mail: mkhurram@ksu.edu.sa).

Ke Gu is with the School of Computer & Communication Engineering, Changsha University of Science & Technology, Changsha 410114, China (e-mail: gk4572@163.com).

Xiaosong Zhang is with the Institute for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China and also with the Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen Guangdong 518040, China (e-mail: johnsonzxs@uestc.edu.cn).

Digital Object Identifier 10.1109/JBHI.2022.3140831

Despite its promising advantages, cloud storage also faces many security challenges. For example, machine downtime that occurred in major cloud service providers has caused a growing number of cloud incidents. For the medical data, the leakage or damage to the integrity of the this data will bring great threats. More importantly, outsourcing medical data to cloud-based medical storage systems (CMSS) means that the patient loses physical control of the data. Therefore, the privacy and integrity protection of the medical data outsourced to medical cloud storage servers become important issues for the patients. In the real world, the security of medical data in the cloud is getting more and more attention [4]. Many countries and institutions have issued their data security regulations and established the regulatory agencies to regulate the safe use of medical data. For example, the European Data Protection Act requires medical data to be approved before being provided to third parties [5].

To address the integrity issue in cloud storage, the concept of data integrity auditing has been proposed, which allows the data owner (DO) to verify the integrity of the outsourced data stored on a cloud storage server (CSS) without downloading all of them. To realize the economies of scale in cloud computing, many third-party auditor (TPA)-based public auditing protocols have been proposed by researchers, which can reduce the DO's computational cost and achieve some ideal features such as batch auditing and dynamic update of data. However, the security and the performance of TPA-based integrity auditing protocols can also be improved. On the security aspect, data breaches via third parties become a common problem in the current security situation [6]. For example, according to the BDO and AusCERT 2018/19 Cyber Security Survey [7], data breaches through third-party providers and suppliers increased by 74.3% in Australia. Therefore, even if TPA is considered trustworthy, the public auditing protocols should ensure the privacy of DO's original data to TPA. On the performance aspect, the communication overhead between TPA and CSS can further be improved. Besides, despite leaving the powerful computing power of CSS aside, the computational efficiency of DO and TPA still needs to be improved, because most existing protocols require a large number of time-consuming bilinear pairing and map-to-point hash operations.

A. Our Contributions

To provide integrity protection for the patients in the CMSS and resolve the above problems, we propose an efficient privacy-preserving public auditing protocol for the CMSS. The contributions of this paper are illustrated as follows:

1) For a CMSS, we summarize the security and functional requirements of the semi-trusted TPA-based public auditing protocol and design an efficient privacy-preserving public auditing protocol in this model.
2) The correctness and security of our protocol are analyzed, and the analysis results show that our protocol achieves the predetermined security and functional goals. In terms of functionality, our protocol supports batch verification and dynamic update of the data. From a security perspective, our protocol can resist various attacks while ensuring storage correctness and can protect the privacy of users' data from being retrieved by TPA.
3) The performance is compared with other related work from theoretical analysis and simulation experiment aspects. The results show that our protocol is more computationally efficient than other related work on TPA and DO sides due to the less use of time-consuming operations such as map-to-point hash and bilinear pairing. Meanwhile, our protocol also has significant advantage in terms of communication cost, and the advantage is more noticeable in batch verification as challenged data blocks increase.

B. Organization

The remainder of the paper is arranged as follows. Section II describes related work about integrity auditing in could storage. Section III introduces some basic knowledge used in the paper. In Section IV, we propose an efficient privacy-preserving public auditing protocol for CMSS. The correctness and security of our protocol are analyzed in Section V, and then we evaluate the performance of our protocol and compare it with that of other related work in Section VI. Finally, we draw our conclusions in Section VII.

II. Related Work

Due to the development of cloud storage, more and more people upload their data to the cloud, and the integrity issue becomes an urgent problem to be solved. In 2007, Ateniese *et al.* [8] proposed the notion of provable data possession (PDP), which uses homomorphic verifiable tags and random sampling techniques to achieve the blockless verification of data possession stored in the untrusted CSS. However, the protocol is only suitable for the integrity auditing of static cloud data and does not support dynamic operations, such as the insertion, deletion, and modification of data blocks. To address this issue, Ateniese *et al.* [9] proposed a scalable and efficient PDP protocol for outsourced cloud data by using symmetric-key cryptography, which first supports the dynamic operations on outsourced data blocks, such as modification and deletion. Later, in 2009, Erway *et al.* [10] proposed a PDP protocol, which uses authenticated dictionaries based on rank information to achieve full dynamic operations, i.e., the modification, deletion, and insertion.

In 2010, to save the cost of data owners, Wang *et al.* [11] first proposed a TPA-based dynamic auditing protocol by combining the Markle Hash Tree and the short signature. The protocol supports public auditing, full dynamic operations, and batch auditing. But it has heavy computational cost and communication overhead during the auditing and update phases. In 2013, Yang and Jia proposed [12] a public auditing protocol for cloud storage, which supports dynamic operation of data and batch verification. However, Ni *et al.* [13] pointed out that Yang and Jia's protocol [12] is vulnerable to a kind of modification attack, and the modified proof can pass the verification of TPA even if the original data has been corrupted. Most TPA-based public auditing protocols have not considered the data leakage problem so that TPA may obtain user's data information in some

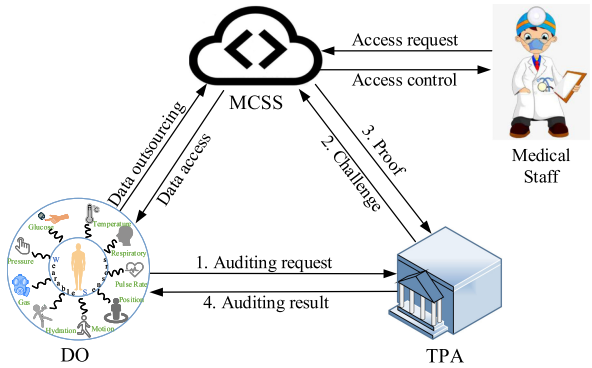


Fig. 1. Model of cloud-based medical storage system.

situations. To address this problem, Wang *et al.* [14] proposed a public auditing protocol with privacy protection for cloud storage by using homomorphic linear signature and random mask technology, such that malicious TPA has no ability to reveal the data of the users.

To improve the efficiency of public auditing protocol, Hao *et al.* [15] proposed an efficient remote data possession checking protocol, which utilizes the doubly linked lists and arrays to improve the efficiency. Recently, Shen *et al.* [16] proposed a new public auditing protocol for cloud data, which is considered efficient due to the adoption of a new storage structure. However, their protocol [16] is vulnerable to the data leakage attack [17], i.e., an adversary, once he is able to compromise TPA, can also recover all users' outsourced data by constructing some suitable challenges. He *et al.* [18] considered the privacy-preserving provable data possession against the data leakage from the verifier. That means TPA could not obtain more information about original data than required. For Cloud-based medical storage system (CMSS), Zhang *et al.* [19] proposed an efficient TPA-based public auditing scheme. Subsequently, many TPA-based public auditing protocols [20]–[24] for cloud storage have been proposed either to improve security and functionality or to enhance efficiency.

III. Preliminaries

This section introduces some preliminaries used in this paper, such as the system model and security requirements of public auditing protocol for CMSS, the cryptography primitive-bilinear pairing, and hash table structure.

A. System Model and Security Requirements

We first describe the system model of the CMSS in this section and then identify the security and functional goals that the auditing protocol should achieve.

1) General System Model: The system model of the CMSS in this paper is shown in Fig. 1, which is a widely accepted system model of TPA-based public auditing protocols. Specifically, there are four entities in the model, i.e., DO (data owner), MCSS (medical cloud storage server), TPA (third-party auditor), and medical staff. What we need to pay attention to is that the medical staff does not participate in the auditing of the medical data, and he/she can access the patients' data in CMSS by the access control mechanism, which is out of the scope of this paper.

1) DO: the data owner, whose data is collected by different medical sensors and outsourced to MCSS, and can access the data as they like. For security concerns, DOs entrust TPA to perform integrity auditing operations of the outsourced data stored on MCSS.
2) MCSS: medical cloud storage server, is an entity that provides medical data storage service for users with large-scale cloud computing infrastructure. Different from traditional servers, MCSS stores massive amounts of medical data containing sensitive information about DOs, so the privacy and integrity protection is especially important. To guarantee the integrity of DOs' data, MCSS accepts the audit of TPA and generates a corresponding proof according to the audit challenge launched by TPA. MCSS is a semi-trusted entity, who performs the corresponding operations according to the protocol but also expects to provide DOs with proof of integrity without storing DOs' original data intact.
3) TPA: a third-party auditor with professional knowledge. According to DO's delegation, TPA sends a integrity audit challenge to MCSS. When receiving the corresponding proof generated by CSS, TPA determines the integrity of the medical data by checking the validity of the proof, and responses the audit result to DO. Generally, TPA is an honest but curious entity, i.e., TPA is curious about the original data of DO even if he/she strictly executes the audit processes.
4) Medical staff: personnel with medical expertise, who can access the patients' medical data stored in the MCSS for health diagnosis and monitoring according to the access control mechanism. Since we only focus on medical data auditing, we omit the detailed introduction of this part.

2) Security and Functional Requirements: Based on previous work in this field, an ideal TPA-based public auditing protocol for CMSS should meet the following requirements:

1) Resistance to forgery attack: Any adversary including MCSS cannot forge auditing proof to pass the verification.
2) Resistance to replay attack: MCSS cannot pass the challenge by replaying previous proofs of the corresponding file.
3) Resistance to replace attack: MCSS cannot generate a valid proof by replacing some challenged data blocks with other data blocks.
4) Privacy-preserving: Any adversary including TPA cannot recover the original file F of DO from the proofs provided by MCSS.
5) Guarantee of storage correctness: DO's original medical data should be stored on MCSS intact, and valid proof should be generated according to the original medical data. Otherwise, MCSS cannot pass the audit of TPA.
6) Batch auditing: To improve the efficiency of protocol execution, TPA can handle multiple challenges from different files simultaneously.

7) Dynamic support: The protocol allows DOs to update (insert, delete, modify, or append) some blocks of data without changing the data of other blocks.
8) High efficiency: The protocol should be computationally efficient to reduce the overhead of TPA and DO, and also should be efficient for TPA and MCSS.

B. Basic Cryptography Knowledge

In this section, we introduce the basic knowledge of used cryptography primitive and security assumptions.

1) Bilinear Pairing: Let G and G_T are two multiplicative cyclic groups with the same order of large prime number p. The map $e : G \times G \to G_T$ is a bilinear pairing [25], [26] if it satisfies the following properties:

Bilinearity: For $g_1, g_2 \in G$ and $a, b \in Z_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.

Non-degeneracy: There exists $g_1, g_2 \in G$ such that $e(g_1, g_2) \neq 1_{G_T}$.

Computability: There is an efficient algorithm to compute $e(g_1, g_2)$ for all $g_1, g_2 \in G$.

2) Security Assumptions: The security of our protocol is based on the following intractable problems.

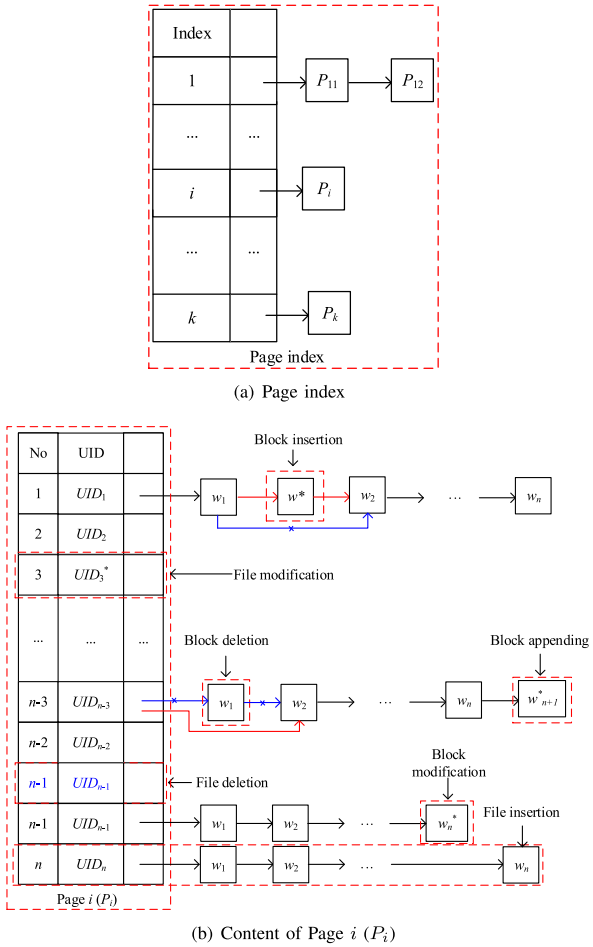
Computational Diffie-Hellman (CDH) Problem: Given g^a and g^b, where g is a generator of group G and a, b are two random numbers in Z_p, compute the value g^{ab}.

Discrete Logarithm (DL) Problem: Given $g^a \in G$, where g is a generator of group G and a is a random number in Z_p, compute a.

C. Hash Table With Dynamic Operations

The improved hash table is a two-dimensional data structure that combines the advantages of array and linked list, which is based on the dynamic hash table given in [27]. It supports fast file search and update operations. So, it can be adopted by TPA in public auditing protocol to store the information related to the files and blocks of DO. By this way, TPA can audit the integrity of the data and support the function of data update. The architecture of the improved hash table is shown in Fig. 2, which contains two components, i.e., the page index and the page content, where the page index identifies which page the file belongs to and the page content stores the actual content related to the file. Firstly, each file is assigned to a page index by hashing the owner's ID and the file ID. As shown in the Fig. 2(a), the page index contains k elements, and which is much smaller than the total number of the files. Besides, each file-related information is stored in the page content, and which is shown in Fig. 2(b). The blocks of a file are stored one by one by using a linked list, and $w_i = k_i \| V_i$ is stored in the list for each block, which contains a random number and a version number of the block. As can be seen from Fig. 2(b), the improved hash table structure supports file operations and block operations, and they are both include modification, deletion, insertion and appending operations. We will describe the dynamic operations of the hash table in conjunction with the protocol in the following section.

To find the location of a file with the improved hash table structure, the page index is first calculated by hashing the owner's ID and file ID to locate the page to which the file belongs. Then, the file could be found by searching the certain page via unique identity UID (owner's ID $\|$ file ID). This structure can enhance the query efficiency, and we compare it with linked list data structure. Suppose that there are n^2 files in the improved hash table, and they are evenly distributed across n different pages with page index $index \in [1, n]$, i.e., each page contains n files. To query each file one time in n^2 files by the improved hash table, we need to query $n^2(n+1)$ times for the page index and the page content. But, if we use a linked list to query each file one time for the n^2 files, we need to query $\frac{n^2 \cdot (n^2+1)}{2}$ times. It is obvious that $\frac{n^2 \cdot (n^2+1)}{2} > n^2(n+1)$, so the query efficiency of the improved hash table is higher than that of the linked list. For example, $n = 10$, 5050 searches are needed for the linked list to query all files one time, while just 1100 searches are required for the improved hash table, and it almost reduces search times by about 78%. Therefore, the hash table of this paper can improve the efficiency of data query.



(a) Page index

(b) Content of Page i (P_i)

Fig. 2. Illustration of hash table supporting dynamic operations.

IV. Protocol Design

In this section, we propose a new public auditing protocol for CMSS under the semi-trusted TPA model, which not only achieves the above security features, but also maintains high

TABLE I
NOTATIONS USED IN THE PAPER

Notation	Description
p	A large prime number
G, G_T	Two multiplicative cyclic groups with order p
g	A generator of G
$h(\cdot)$	A secure hash function from $\{0,1\}^*$ to Z_p
$e: G \times G \rightarrow G_T$	A bilinear pairing
(K_S, K_P)	MCSS's secret/public keys for signature
F	DO's file with n blocks $\{m_1, m_2, \cdots, m_n\}$
m_i	The ith block of a certain data file
ID	The unique identity of DO
FID	The unique identity (name) of a DO's file
SK, PK	A private and public key sets of DO
t_i	Tag of the ith block generated by DO
$SIG_{sk}(\cdot)$	Signature with private key sk
V_i	Current version of the ith block
k_i	Random number for the ith block
T	Set of $\{t_i\}_{i\in[1,n]}$

computational and communication efficiency. The notations used in the protocol are shown in the Table I.

A. The Description of Our Protocol

The proposed protocol contains two phases, i.e., the setup phase and the verification phase, and we describe the two phases of our protocol as below.

1) Setup Phase: This phase contains three algorithms, i.e., **KeyGen**, **TagGen** and **HTGen**. Through these algorithms, DO outsources the data files to MCSS and stores the information related to data auditing in TPA.

KeyGen: DO runs this algorithm to generate private and public key pairs. DO generates the key pair $(SK = \{sk, x_1, x_2\}, PK = \{pk, y_1, y_2\})$, where (sk, pk) is a private and public key pair for signature, x_1, x_2 are two random private keys chosen from Z_p^*, and $y_1 = g^{x_1}$ and $y_2 = g^{x_2}$ are two corresponding public keys. Then, DO keeps private keys $SK = \{sk, x_1, x_2\}$ secretly, and publishes the public keys $PK = \{pk, y_1, y_2\}$.

TagGen: When the data is collected by different medical sensors, DO first generates the tags corresponding to the outsourced data blocks by running the TagGen algorithm, and sends the data with the tags to MCSS. For a file F collected by the medical sensor, DO separates it to n blocks $F = \{m_1, m_2, \ldots, m_n\}$. Then, DO computes a tag for each block $t_i = g^{x_1(x_2 \cdot m_i + h(w_i))}$ $(i \in [1, n])$, where $w_i = k_i \| V_i$, k_i is a random number chosen by DO for the ith data block, and V_i is the current version number of the ith data block. The n tags form a tag set $T = \{t_i\}_{i\in[1,n]}$. Besides, DO computes a file tag $\theta = (ID \| FID \| n) \| SIG_{sk}(ID \| FID \| n)$, where $SIG_{sk}(ID \| FID \| n)$ represents the signature of $(ID \| FID \| n)$ by using sk. Finally, DO uploads $\{F, T, \theta\}$ to MCSS for storage. Here we should note that for the medical staff, if the access control mechanism of the file is satisfied, he/she can access the file F.

HTGen: A hash table (HT) of the data information is generated by TPA through running this algorithm, which will be used for challenge and verification of the data auditing. According to $w_i = k_i \| V_i (i \in [1, n])$, DO forms an information set $W = \{w_i\}_{i\in[1,n]}$, and submits $\{(ID \| FID \| n), W)\}$ to TPA. Upon receiving the data information, TPA adds it to the HT according to the introduction in Section III-C.

2) Verification Phase: This phase is also composed of three algorithms, i.e., **Challenge**, **ProofGen** and **ProofVer**. According to DO's delegation, TPA runs **Challenge** algorithm to randomly generate a challenge of the file for MCSS. After receiving the challenge, MCSS generates a proof by running **ProofGen** algorithm according to the challenged blocks, and returns it to TPA for integrity verification. Then TPA verifies the integrity of data by running **ProofVer** algorithm, and returns the result to DO.

Challenge: DO submits $\{ID \| FID \| n\}$ to TPA as a delegation of an auditing. TPA requests MCSS for the corresponding file tag θ, and verifies the signature $SIG_{sk}(ID \| FID \| n)$ by using pk. If the verification fails, TPA notifies DO that the file is corrupted. Otherwise, the process continues. TPA randomly selects c data blocks $IDX = \{j\}_{j\in[1,c]}$ for challenge, and generates a random number $r \in Z_p^*$. Then TPA sends a challenge $chal = \{IDX = \{j\}_{j\in[1,c]}, r\}$ to MCSS.

ProofGen: Upon receiving the challenge from TPA, MCSS computes $SUM = \sum_{j\in[1,c]} h(r, j) \cdot m_j \bmod p$, $PD = y_2^{SUM}$, $PT = \prod_{j\in[1,c]} t_j^{h(r,j)}$, $PU = e(PD, y_1)$ and $PV = e(PT, g)$ according to the challenged data blocks. Then, MCSS signs the $\{PU, PV\}$ using K_S as $SIG_{K_S}(h(PU, PV))$, and returns the corresponding proof $\{PU, PV, SIG_{K_S}(h(PU, PV))\}$ back to TPA.

ProofVer: When getting the proof returned by MCSS, TPA first checks the validity of the signature $SIG_{K_S}(h(PU, PV))$ by using the public key K_P. The session is terminated if the signature verification fails. Otherwise, TPA retrieves $\{w_j\}_{j\in[1,c]}$ from the hash table according to the challenged blocks. Then, TPA computes $\nu = \sum_{j\in[1,c]} h(r, j) \cdot h(w_j) \bmod p$, and checks $PU \cdot e(g^{\nu}, y_1) \stackrel{?}{=} PV$. If they are equal, the integrity of the outsourced data is verified by TPA. Otherwise, the integrity of the outsourced data is corrupted. Finally, TPA returns the result of the data audit to DO.

B. Dynamic Operations

Dynamic update of the data is a very practical function for DO, which makes the auditing protocol act normally when DO updates the data stored in MCSS. In our protocol, the hash table structure is used to realize dynamic operations on the outsourced data, such as insertion, deletion, modification and append operations. Since the file-level dynamic operations are based on the block-level operations, we just introduce the dynamic operations of blocks.

1) Block Insertion: Assuming that DO wants to insert a new block m^* after the ith block m_i for a file. In order to update the hash table and achieve the insertion operation, DO runs the textbfTagGen algorithm for the new data block to generate the corresponding tag t^*. Firstly, DO generates a random number k^*, and calculates $t^* = g^{x_1(x_2 \cdot m^* + h(w^*))}$, where $w^* = k^* \| V^*$, and $V^* = 1$. Then, DO sends the dynamic insertion request $B_{Insert2C} = \{\text{Insert}, ID, FID, m^*, t^*, i\}$ to the MCSS, and MCSS updates the data according to the insertion request. Meanwhile, DO sends the insertion request $B_{Insert2T} =$

$\{\text{Insert}, ID, FID, w^*, i\}$ to TPA. When receiving the request, TPA finds the location of the file by using $ID\|FID$ as shown in III-C, and then finds the ith block. Finally, DO inserts the new data block information w^* behind the ith block.

2) Block Deletion: Assuming that DO wants to delete the ith block of a file, DO sends the block deletion request $B_{Delete2C} = \{\text{Delete}, ID, FID, i\}$ to MCSS. When getting the request, MCSS deletes the corresponding m_i and t_i from the database data according to ID and FID. Meanwhile, DO sends the deletion request $B_{Delete2T} = \{\text{Delete}, ID, FID, i\}$ to TPA. When receiving the request, TPA finds the location of the file by using $ID\|FID$ as shown in III-C, and finds the ith block. Then, TPA deletes w_i from the hash table, and connects w_{i-1} and w_{i+1} together.

3) Block Modification: Assuming that DO wants to modify the ith block m_i of a file to m_i^{new}. DO runs the TagGen algorithm to generate a new tag t_i^{new}. Firstly, DO generates a new random number k_i^{new}, and runs the TagGen algorithm to calculate a new tag $t_i^{new} = g^{x_1(x_2 \cdot m_i^{new} + h(w_i^{new}))}$, where $w_i^{new} = k_i^{new} \| V_i^{new}$ and $V_i^{new} = V_i + 1$. DO sends the block modification request $B_{Modify2C} = \{\text{Modify}, ID, FID, i, m_i^{new}, t_i^{new}\}$ to MCSS. Accordingly, MCSS updates the modified block as the block modification request. Meanwhile, DO sends the update information of the block modification request $B_{Modify2T} = \{\text{Modify}, ID, FID, i, w_i^{new}\}$ to TPA. When receiving the request, TPA finds the location of the file by using $ID\|FID$ as shown in III-C and the ith block. At last, TPA replaces w_i with w_i^{new}.

4) Block Appending: Assuming that DO wants to append a new block m^* to the end of a file with n blocks, DO runs the TagGen algorithm for the new data block to generate the corresponding tag t^*. Firstly, DO generates a random number k^*, and calculates $t^* = g^{x_1(x_2 \cdot m^* + h(w^*))}$, where $w^* = k^* \| V^*$, and $V^* = 1$. Besides, DO signs $ID\|FID\|n+1$ as $SIG_{sk}(ID\|FID\|n+1)$ by using sk. Then, DO sends a request $B_{Append2C} = \{\text{Append}, ID, FID, m^*, t^*\}$ to the MCSS, and MCSS appends the $(n+1)$th block m^* behind the nth block. Meanwhile, DO sends a request $B_{Append2T} = \{\text{Append}, ID, FID, w^*\}$ and $SIG_{sk}(ID\|FID\|n+1)$ to TPA. When receiving the request, TPA finds the location of the file by using $ID\|FID$ as shown in III-C, and appends the $(n+1)$th w^* behind the nth w_n. Besides, TPA updates the signature to the new one $SIG_{sk}(ID\|FID\|n+1)$.

C. Batch Auditing

In addition to the basic function of single file auditing, the protocol also supports batch auditing for multiple files from a DO or multiple DOs, which can greatly improve the efficiency of auditing. The batch auditing of our protocol also contains three algorithms: **BChallenge**, **BProofGen** and **BProofVer**.

Suppose that DO wants to check the integrity of b files $\{FID_1, FID_2, \ldots, FID_b\}$, the specific workflow of the batch auditing is as follows:

BChallenge: DO submits $\{ID\|FID_k\|n_k\}_{k\in[1,b]}$ to TPA as a delegation of the batch auditing. TPA requests MCSS for the corresponding tags of these files $\{\theta_k\}_{k\in[1,b]}$, and verifies the signatures $\{SIG_{sk}(ID\|FID_k\|n_k)\}_{k\in[1,b]}$ by using pk. If one of the verifications fails, TPA notifies DO that at least one of the file is corrupted. Otherwise, the process continues. TPA randomly selects c data blocks $IDX_k = \{kj\}_{j\in[1,c]}$ for file FID_k, and generates a random number $r \in Z_p$. Then TPA sends the challenge $chal = \{\{IDX_k = \{kj\}_{j\in[1,c])}\}_{k\in[1,b]}, r\}$ to MCSS.

BProofGen: Upon receiving the challenge from TPA, MCSS computes $SUM = \sum_{k\in[1,b]} \sum_{j\in[1,c]} h(r, kj) \cdot m_{kj} \mod p$, $PD = y_2^{SUM}$, $PT = \prod_{k\in[1,b]} \prod_{j\in[1,c]} t_{kj}^{h(r,kj)}$, $PU = e(PD, y_1)$ and $PV = e(PT, g)$ according to the challenged data blocks. Then, MCSS signs the $\{PU, PV\}$ using K_S as $SIG_{K_S}(h(PU, PV))$, and returns the corresponding proof $\{PU, PV, SIG_{K_S}(h(PU, PV))$ back to TPA.

BProofVer: When getting the proof from MCSS, TPA first checks the validity of the signature $SIG_{K_S}(h(PU, PV))$ by using the public key K_P. The session is terminated if the signature verification fails. Otherwise, TPA retrieves $\{w_{kj}\}_{(k\in[1,b], j\in[1,c])}$ from the hash table according to the challenged files and blocks. Then, TPA computes $\nu = \sum_{k\in[1,b]} \sum_{j\in[1,c]} h(r, kj) \cdot h(w_{kj}) \mod p$, and checks $PU \cdot e(g^{\nu}, y_1) \stackrel{?}{=} PV$. If they are equal, the integrity of the outsourced files is verified by TPA. Otherwise, the integrity of part outsourced files is corrupted. Finally, TPA returns the result of the batch auditing to DO.

V. Correctness and Security Analysis

In this section, we first give the correctness analysis of the proposed protocol. Then we prove that our protocol is secure and achieves the predetermined goals.

A. Correctness Analysis

The correctness of the protocol means that it can complete the data integrity audit well, that is, the proof generated according to the challenge data blocks can pass the data integrity check. We analyze the correctness of our protocol for both the basic protocol and the batch verification.

The correctness analysis of the basic protocol based on the judgment formula is as follows. In the left formula, PU is a part of proof from MCSS, and v is computed by TPA according to the information related to the challenged blocks. In the right formula, PV is another part of proof and it is calculated from $\{t_j\}_{j\in[1,n]}$.

$$
\begin{aligned}
& PU \cdot e(g^{\nu}, y_1) \\
&= e(PD, y_1) \cdot e\left(g^{\sum_{j\in[1,c]} h(r,j)\cdot h(w_j)}, y_1\right) \\
&= e(y_2^{SUM}, y_1) \cdot e\left(g^{\sum_{j\in[1,c]} h(r,j)\cdot h(w_j)}, y_1\right) \\
&= e\left(g^{x_2 \cdot \sum_{j\in[1,c]} h(r,j)\cdot m_j}, y_1\right) \cdot e\left(g^{\sum_{j\in[1,c]} h(r,j)\cdot h(w_j)}, y_1\right) \\
&= e\left(\prod_{j\in[1,c]} \left(g^{x_1\cdot(x_2\cdot m_j + h(w_j))}\right)^{h(r,j)}, g\right)
\end{aligned}
$$

$$= e\left(\prod_{j\in[1,c]} t_j^{h(r,j)}, g\right)$$
$$= e(PT, g)$$
$$= PV$$

From the above equation, we can clearly see that the correctness of the integrity auditing for basic protocol is established.

The correctness analysis of the batch auditing is as follows. In the left formula, PU is also a part of proof from MCSS. But, different from basic protocol, PU in the batch verification is calculated from blocks of multiple files. v is computed by TPA according to the information related to the challenged blocks in the batch files. In the right formula, PV is another part of proof and it is calculated from $\{t_{kj}\}_{k\in[1,b],j\in[1,c]}$.

$$PU \cdot e(g^{\nu}, y_1)$$
$$= e(PD, y_1) \cdot e\left(g^{\sum_{k\in[1,b]}\sum_{j\in[1,c]} h(r,kj)\cdot h(w_{kj})}, y_1\right)$$
$$= e(y_2^{SUM}, y_1) \cdot e\left(g^{\sum_{k\in[1,b]}\sum_{j\in[1,c]} h(r,kj)\cdot h(w_{kj})}, y_1\right)$$
$$= e\left(g^{x_2\cdot\sum_{k\in[1,b]}\sum_{j\in[1,c]} h(r,kj)\cdot m_{kj}}, y_1\right)$$
$$\cdot e\left(g^{\sum_{k\in[1,b]}\sum_{j\in[1,c]} h(r,kj)\cdot h(w_{kj})}, y_1\right)$$
$$= e\left(g^{\sum_{k\in[1,b]}\sum_{j\in[1,c]} x_2\cdot h(r,kj)\cdot m_{kj} + h(r,kj)\cdot h(w_{kj})}, g^{x_1}\right)$$
$$= e\left(g^{\sum_{k\in[1,b]}\sum_{j\in[1,c]} h(r,kj)\cdot (x_2\cdot m_{kj} + h(w_{kj}))}, g^{x_1}\right)$$
$$= e\left(g^{\sum_{k\in[1,b]}\sum_{j\in[1,c]} h(r,kj)\cdot x_1\cdot (x_2\cdot m_{kj} + h(w_{kj}))}, g\right)$$
$$= e\left(\prod_{k\in[1,b]}\prod_{j\in[1,c]} \left(g^{x_1\cdot(x_2\cdot m_{kj}+h(w_{kj}))}\right)^{h(r,kj)}, g\right)$$
$$= e\left(\prod_{k\in[1,b]}\prod_{j\in[1,c]} t_{kj}^{h(r,kj)}, g\right)$$
$$= e(PT, g)$$
$$= PV$$

From the above equation, we can clearly see that the correctness of the butch auditing is established, i.e., our protocol can verify the integrity of batch files at one time.

B. Security Proof

1) Storage Correctness: Assuming that the CDH problem and DL problem in G are intractable, the proposed protocol guarantees the feature of storage correctness, i.e., if the MCSS does not store DO's data intact, it cannot forge a proof to pass the verification of TPA.

Proof: The proof is based on the proof method in [28]. If the adversary $\mathcal{A}$ (MCSS) can generate a proof to pass the verification of TPA without storing the data of DO intact, the challenger $\mathcal{C}$ (act as DO) can construct a simulator to solve the CDH or DL problems. The proof is accomplished with the following games.

Game 1: Both $\mathcal{C}$ and $\mathcal{A}$ normally act as the description in Section IV-A. That is, $\mathcal{C}$ runs the **KeyGen** to obtain $\{SK, PK\}$, and sends the public parameters PK to $\mathcal{A}$. $\mathcal{A}$ queries the signatures of data blocks. $\mathcal{C}$ runs **TagGen** to generates the tag set $T = \{t_i\}_{i\in[1,n]}$ and file tag θ, and sends the tag set and file tag to $\mathcal{A}$. Then $\mathcal{C}$ sends a challenge to $\mathcal{A}$. Finally, $\mathcal{A}$ returns the proof $\{PU, PV\}$ to the challenger.

Game 2: Game 2 is similar to the Game 1, with one difference. When receiving the challenge from $\mathcal{C}$, $\mathcal{A}$ responses a forged proof $\{PU,' PV'\}$, which is different from the expected proof $\{PU, PV\}$. If the forged proof can pass the verification, $\mathcal{A}$ wins the game.

Analysis: Assuming that $\mathcal{A}$ wins the Game 2 with non-negligible probability. Then we can construct a simulator to solve the CDH problem, i.e. knowing $y_1 = g^{x_1}$ and $y_2 = g^{x_2}$, it can calculate $g^{x_1x_2}$ without using x_1 and x_2.

Assuming that the correct proof from the honest prover is $\{PU, PV\}$, where $PU = e(PD, y_1)$, $PD = y_2^{SUM}$ and $PV = e(PT, g)$, and it meets the verification equation $PU \cdot e(g^v, y_1) = PV$, i.e. $e(y_2^{SUM}, y_1) \cdot e(g^v, y_1) = e(PT, g)$.

Assuming that the proof $\{PU,' PV'\}$ forged by $\mathcal{A}$ can pass the verification, where $PU' = e(PD,' y_1)$, $PD = y_2^{SUM'}$ and $PV = e(PT,' g)$. Then, $PU' \cdot e(g^v, y_1) = PV'$, i.e. $e(y_2^{SUM'}, y_1) \cdot e(g^v, y_1) = e(PT,' g)$.

By dividing the above equations, we can get $e(y_2^{SUM-SUM'}, y_1) = e(PT \cdot PT'^{-1}, g)$. There is at least a data block $m_i' \neq m_i$ such that $SUM \neq SUM'$, otherwise $PU = PU'$ and $PV = PV'$. Assuming that $\Delta SUM = SUM - SUM'$. As a result, we have $e(y_2^{SUM-SUM'}, y_1) = e(y_2^{\Delta SUM}, y_1) = e(g^{x_1x_2\Delta SUM}, g) = e(PT \cdot PT'^{-1}, g)$. Therefore, $g^{x_1x_2\Delta SUM} = PT \cdot PT'^{-1}$, and $g^{x_1x_2}$ can be calculated as $g^{x_1x_2} = (PT \cdot PT'^{-1})^{\frac{1}{\Delta SUM}}$ without using x_1 and x_2. This contradicts with the assumption that the CDH problem in G is intractable.

Game 3: Game 3 is similar to the Game 2, the challenger $\mathcal{C}$ chooses c blocks to challenge the adversary $\mathcal{A}$ by sending $chal = \{IDX = \{j\}_{j\in[1,c]}, r\}$. And then, the adversary $\mathcal{A}$ computes the proof $\{PU,' PV\}$, where $PU' = e(PD,' y_1)$, $PD' = y_2^{SUM'}$, and SUM' is not equal to the expected SUM. If $PU' \cdot e(g^v, y_1) = PV$, we say that the adversary $\mathcal{A}$ wins the game.

Analysis: Assuming that the adversary wins the Game 3 with non-negligible probability. Then we can construct a simulator to solve the DL problem. Given $(g, w) \in G$, the goal of the simulator is to calculate value α such that $w = g^{\alpha}$. The simulator acts as $\mathcal{C}$ in Game 3, but with some differences.

In the Setup phase, the simulator generates a random number x_1 from Z_p, and calculates the corresponding public key $y_1 = g^{x_1}$. Then, the simulator chooses two random numbers $a, b \in Z_p$, and sets $y_2 = g^a w^b$. The tag of each block can be calculated as $t_i = y_2^{x_1 m_i} g^{x_1 h(w_i)}$ $(i \in [1, n])$.

Assuming that the correct proof from the honest prover is $\{PU, PV\}$, where $PU = e(PD, y_1)$ and $PD = y_2^{SUM}$, and it meets the verification equation $PU \cdot e(g^v, y_1) = PV$, i.e. $e(y_2^{SUM}, y_1) \cdot e(g^v, y_1) = PV$.

Assuming that the proof $\{PU,' PV\}$ forged by $\mathcal{A}$ can also pass the verification, and we have $PU' \cdot e(g^v, y_1) = PV$, i.e. $e(y_2^{SUM'}, y_1) \cdot e(g^v, y_1) = PV$.

From the above two equations, we have $e(y_2^{SUM}, y_1) = e(y_2^{SUM'}, y_1)$, and then we can get $y_2^{SUM} = y_2^{SUM'}$. There is at least a data block $m_i' \neq m_i$ such that $SUM \neq SUM'$. Otherwise, $PU = PU'$. Let $\Delta SUM = SUM' - SUM \neq 0$, we have $1 = y_2^{\Delta SUM} = (g^a w^b)^{\Delta SUM} = g^{a\Delta SUM} \cdot w^{b\Delta SUM}$. So the solution of the DL problem can be found as $w = g^{-\frac{a\Delta SUM}{b\Delta SUM}} = g^{-\frac{a}{b}}$. We should note that the probability of $b = 0$ is $\frac{1}{p}$. In other words, the probability that we find a solution of the DL problem is $1 - \frac{1}{p}$, which contradicts with the assumption that the DL problem in G is intractable.

Based on the above analysis, due to the difficulty of CDH problem and DL problem in G, the MCSS cannot forge the proof to pass the verification when it does not store the original data intact. In other words, our protocol ensures the feature of storage correctness.

2) Resistance to Forgery Attack: As shown in the analysis of the storage correctness part, the MCSS cannot forge the proof to pass the verification without completely storing DO's original data. Besides, in the **ProofGen** algorithm of our protocol, in addition to the $\{PU, PV\}$, the signature $SIG_{K_S}(h(PU, PV))$ is also contained in the proof. Therefore, without knowing MCSS's private key K_S for the signatures, any adversary containing the malicious TPA cannot forge the proof to pass the verification. In other words, our protocol can resist forgery attack.

3) Resistance to Replay Attack: In each challenge phase of the proposed protocol, TPA randomly picks c blocks and random number r as the challenge, which ensures the freshness of each challenge. In each challenge, PD is calculated according to the original data m_j corresponding to the challenged blocks and the random numbers $h(r, j), j \in [1, c]$, while PT is calculated according to the tags corresponding to the challenged blocks and the random numbers $h(r, j), j \in [1, c]$. Then the proof (PU, PV) can be generated according to (PD, PT). The proof of the current challenge can only be generated based on the challenge information. Therefore, MCSS cannot replay previous proofs for the new challenge, and our protocol can resist replay attack.

4) Resistance to Replacing Attack: MCSS may want to perform replacing attack and hope to generate a proof to pass the verification by replacing the challenged block i with an unchallenged block k. In order to perform this attack, when receiving the challenge $chal = \{IDX = \{j\}_{j\in[1,c]}, r\}$ from TPA, MCSS computes $SUM' = \sum_{j\in[1,c]\&j\neq i} h(r, j) \cdot m_j + h(r, i) \cdot m_k$, $PD' = y_2^{SUM'}$, $PT' = \prod_{j\in[1,c]\&j\neq i} t_j^{h(r,j)} \cdot t_k^{h(r,i)}$, $PU' = e(PD,' y_1)$, and $PV' = e(PT,' g)$. Then, MCSS signs the $\{PU,' PV'\}$ using K_S as $SIG_{K_S}(PU,' PV')$, and returns the corresponding proof $\{PU,' PV,' SIG_{K_S}(PU,' PV')\}$ back to TPA. TPA checks the validity of the proof, and the replacing attack succeeds if the proof passes the verification. Otherwise, the replacing attack fails.

According to the forged proof, the left part of the verification is

$$\begin{aligned} PU' \cdot e(g^{\nu}, y_1) &= e(PD,' y_1) \cdot e(g^{\sum_{j\in[1,c]} h(r,j)\cdot h(w_j)}, y_1) \\ &= e(y_2^{SUM'}, y_1) \cdot e(g^{\sum_{j\in[1,c]} h(r,j)\cdot h(w_j)}, y_1) \\ &= e(g^{(\sum_{j\in[1,c]\&j\neq i} x_2\cdot h(r,j)\cdot m_j + x_2\cdot h(r,i)\cdot m_k)}, g^{x_1}) \\ &\quad \cdot e(g^{\sum_{j\in[1,c]} h(r,j)\cdot h(w_j)}, g^{x_1}) \\ &= e(g^{\sum_{j\in[1,c]\&j\neq i} x_1(x_2\cdot m_j + h(w_j))\cdot h(r,j)}, g) \\ &\quad \cdot e(g^{x_1(x_2\cdot m_k + h(w_i))\cdot h(r,i)}, g) \\ &= e(\prod_{j\in[1,c]\&j\neq i} g^{x_1(x_2\cdot m_j + h(w_j))\cdot h(r,j)}, g) \\ &\quad \cdot e(g^{x_1(x_2\cdot m_k + h(w_i))\cdot h(r,i)}, g) \end{aligned}$$

The right part of the verification is

$$\begin{aligned} PV' &= e(PT,' g) = e(\prod_{j\in[1,c]\&j\neq i} t_j^{h(r,j)} \cdot t_k^{h(r,i)}, g) \\ &= e(\prod_{j\in[1,c]\&j\neq i} g^{x_1(x_2\cdot m_j + h(w_j))\cdot h(r,j)}, g) \\ &\quad \cdot e(g^{x_1(x_2\cdot m_k + h(w_k))\cdot h(r,i)}, g) \end{aligned}$$

By comparing the left part with the right part, we can know that they will be equal only when $h(w_i) = h(w_k)$. However, according to its construction, w_i is unique for each block and not equal to w_k. Therefore, the proof generated by the above method cannot pass the verification, and our protocol can resist replacing attack.

5) Privacy-Preserving: In Shen *et al.*'s protocol [16], the data proof $D = \prod_{i\in[1,s]} m_i \cdot r_i$ is a linear combination of random numbers and data blocks, and transmitted via a public channel. Therefore, any adversary who compromises TPA, can recover the original data of DO by constructing appropriate challenges [17]. However, in our protocol, the proof is $(PU, PV, SIG_{K_S}(PU, PV)$, where $PU = e(PD, y_1), PV = e(PT, g)$, $PD = y_2^{\sum_{j\in[1,c]} h(r,j)\cdot m_j}$, and $PT = \prod_{j\in[1,c]} t_j^{h(r,j)}$. The data blocks are protected by the bilinear pairing, and TPA has no way to recover DOs' original data from the proof due to the intractability of the discrete logarithm in G_T. Therefore, our protocol can guarantee the privacy of DOs' data.

6) Sampling Audit and MCSS Misbehavior Detection: From a technical point of view, the malicious behavior of MCSS can be prevented if TPA audits all the blocks of the file. However, it will consume more computational costs. To balance the computational efficiency and misbehavior detection of MCSS, the "sampling" technique is first introduced by [8], which greatly reduces the workload of MCSS, while detecting the malicious behavior of MCSS with high probability. According to the analysis of [8], when 1% of data blocks are deleted by MCSS, TPA just needs to challenge 300 blocks and 460 blocks to detect the misbehavior of MCSS with probability 95% and 99%, respectively. Our protocol adopts the "sampling" technique, and can also balance the computational efficiency and misbehavior detection of MCSS.

VI. Comparisons With Related Protocols

In this section, we compare our protocol with other three related protocols [16], [23], [24] in three aspects, namely security and functional features, computational cost and communication overhead.

A. Security and Functionality Comparison

We use the security and functional requirements listed in Section III-A2 as the evaluation criteria for the comparison. According to previous work, we list the results of the comparison in

TABLE II
SECURITY AND FUNCTIONAL COMPARISON

	Our Protocol	[16]	[24]	[23]
Resistance to forgery attack	√	√	√	√
Resistance to replay attack	√	√	√	√
Resistance to replace attack	√	√	√	√
Privacy preserving	√	×	√	√
Storage correctness	√	√	√	√
Batch verification	√	√	×	×
Dynamic support	√	√	√	√
Semi-trusted TPA	√	×	√	×

TABLE III
NOTATIONS FOR COMPUTATIONAL COST EVALUATION

Notation	Description
T_p	Time for a bilinear pairing operation
T_{eZ}	Time for exponent operation on Z_r
T_{eG}	Time for a exponent operation on G_1, G_2 or G
T_{eG_T}	Time for a exponent operation on G_T
T_{mG}	Time for a multiplication operation in G
T_{mG_T}	Time for a multiplication operation in G_T
T_{mZ}	Time for a multiplication operation in Z_p
T_s	Time for generating or verifying a signature
T_{h2G}	Time for a map-to-point hash operation
T_h	Time for a hash operation $h(\cdot)$

Table II. As we can see from the table, Shen *et al.*'s protocols [16] cannot guarantee the feature of privacy-preserving. Actually, Shen *et al.*'s protocol [16] is vulnerable to data privacy breach attack [17], and an adversary may obtain all users' outsourced data once he compromises TPA. Besides, Shen *et al.*'s protocol [16] and Guo *et al.*'s protocol [23] both rely on the trusted TPA, this assumption weakens the scope of the protocols in practical applications. On the one hand, it is difficult to find a trusted third party in reality. On the other hand, data leaks caused by third parties are becoming increasingly frequent. While in terms of functionality, both Guo *et al.*'s protocol [23] and Hahn *et al.*'s protocol [24] do not support batch verification of multiple files. Compared with the related work in [16], [23], [24], our protocol meets all the predetermined security and functional requirements that defined in Section III-A. Therefore, our protocol is more secure in real-world applications.

B. Computational Cost Comparison

For the computational cost comparison, we comprehensively compare the efficiency of three algorithms: tag generation algorithm (DO), proof generation algorithm (MCSS/CSS) and verification algorithm (TPA). In the comparison, we assume that each file contains n blocks, and c blocks are challenged each time. Besides, each block contains s sectors in Guo *et al.*'s protocol [23]. In order to facilitate the computational cost comparison, some notations are defined in Table III. Among these operations, T_{h2G} is the most time-consuming one, followed by T_{eG}, T_{mG} and T_p. The computational complexity of other operations is much lower than that of these four operations, and do not exceed than 1 ms.

In our protocol, DO requires $nT_{eG} + 2nT_{mZ} + T_s + nT_h$ to generate n tags for a file. In the proof generation process, the computational overhead required for MCSS to generate a proof is $2T_p + (c+1)T_{eG} + (c-1)T_{mG} + cT_{mZ} + T_s + (c+1)T_h$. Subsequently, the computational overhead required by TPA is $T_p + T_{eG} + T_{mG_T} + cT_{mZ} + 2T_s + (2c+1)T_h$ to verify the validity of the proof.

In Shen *et al.*'s protocol [16], the calculation overhead required for DO to generate n tags for a file is $2nT_{eG} + nT_{mG} + T_s + nT_{h2G}$. In the proof generation process, CSS needs $cT_{eG} + (c-1)T_{mG} + cT_{mZ}$ to generate a proof to respond to a challenge. Subsequently, to verify the validity of the proof, the computational overhead required by TPA is $(c+2)T_p + T_{eG} + cT_{mG_T} + T_s + cT_{h2G}$.

In Guo *et al.*'s protocol [23], each block is further divided into s sectors. To generate n tags for a file, DO requires a computational overhead of $3nT_{eG} + nT_{mG} + nsT_{mz} + nT_h$. In the proof generation process, the computational overhead required for CSS to generate a proof is $2csT_{mz} + 2cT_{eG} + (c-1)T_{mG}$. Subsequently, the computational overhead required by TPA for the proof verification is $2T_p + (c+s)T_{mG} + (2+s)T_{eG}$.

In Hahn *et al.*'s protocol [24], the calculation overhead for algorithm $SigGen(\cdot)$ required for DO to generate n tags of a file is $nT_h + nT_h + 2nT_{mZ} + 2(n-1)T_{eZ} + (2n+1)T_{h2G} + nT_{mG} + nT_{eG}$. In the proof generation process, CSS needs $nT_h + cT_{mZ} + cT_{eG} + cT_{mZ}$ to generate a proof as a response to the challenge. Then, to verify the validity of the proof, the computational overhead required by TPA is $(c+2)T_{h2G} + 2cT_{eG} + 2(c-1)T_{mG} + 2T_p + 2T_{mG}$.

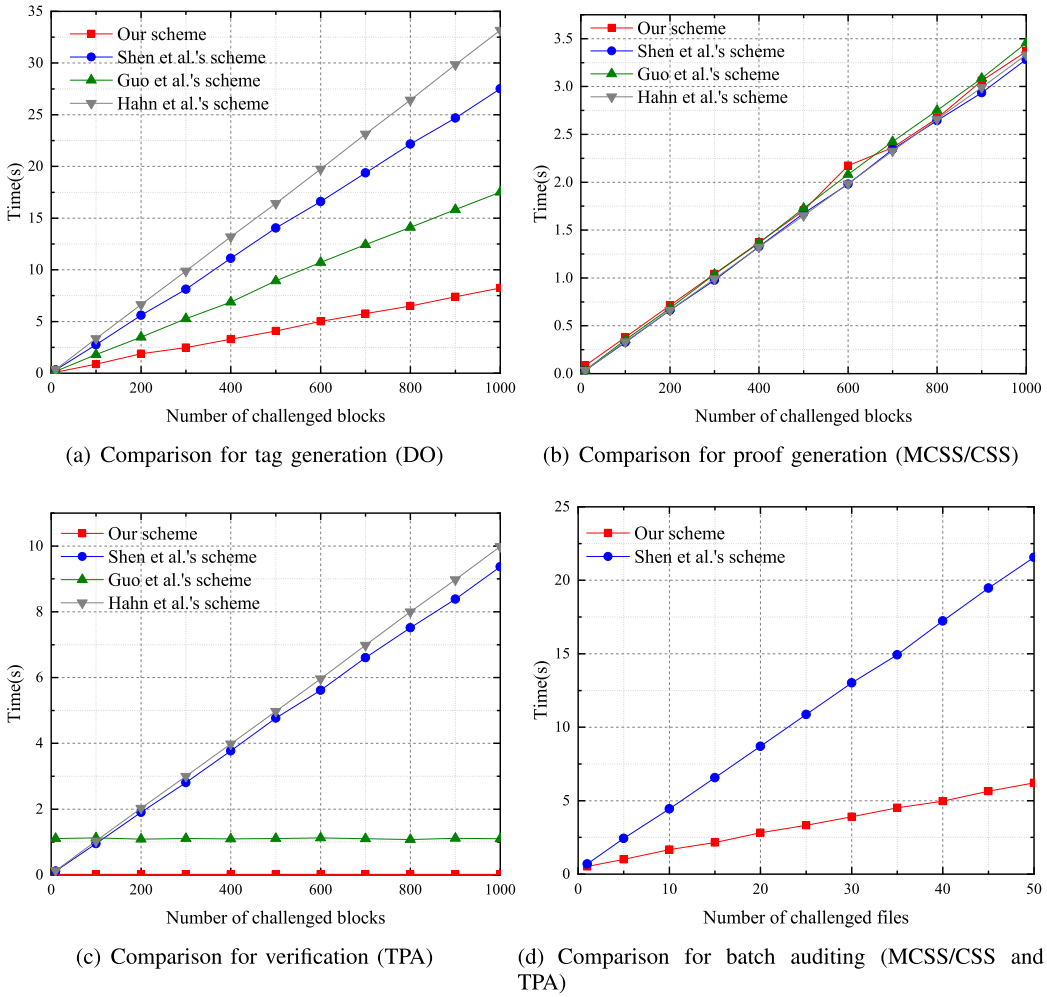
We list the above analyzed computational cost of four protocols in the Table IV. From the table, it is easy to see that the computational cost of our protocol on DO side is less than other three protocols [16], [23], [24] since our protocol needs less T_{eG} operations and does not need the time-consuming operations of map-to-point hash T_{h2G} or multiplication T_{mG}. Meanwhile, compared with the other three protocols [16], [23], [24], our protocol significantly reduces the computational cost of TPA, because the integrity audit of our protocol does not need to perform map-to-point hash and only requires 1 bilinear pairing operation.

In order to show the computational efficiency of our protocol and other related protocols [12], [16], [23] more intuitively, in addition to above theoretical analysis, we also simulate the protocols using the Java language on the Intellij IDEA platform based on the jpbc 2.0.0 library, and the simulations are performed under i5 8400 @ 2.80 GHz processor and 16 G memory with Windows 10 system. Besides, the type A pairing (a.properties) is used in the experiment. Furthermore, the file F is divided into n data blocks in the setting, and each of which is 16 kb. Besides, the block is further divided into $s = 128$ sectors in protocol [23]. The number of challenged data blocks is defined as 40% of the total data blocks. To keep the fairness of the experiments, the simulations of different protocols are made under the same operation environment with the same size of files/blocks and the same system parameters.

The experimental results are shown in Fig. 3, where we simulated the computational cost of tag generation process (DO), the proof generation (MCSS/CSS), the proof verification (TPA) and the batch auditing (MCSS and TPA). For the computational cost of DO to generate file tags, Fig. 3(a) shows that the computational efficiency of our protocol is much better than the protocols

TABLE IV
COMPUTATIONAL COST COMPARISON

	Our Protocol	Protocol in [16]	Protocol in [24]	Protocol in [23]
Computational cost on DO	$nT_{eG} + 2nT_{mZ} + T_s + nT_h$	$2nT_{eG} + nT_{mG} + T_s + nT_{h2G}$	$nT_h + nT_h + 2nT_{mZ} + 2(n-1)T_{eZ} + (2n+1)T_{h2G} + nT_{mG} + nT_{eG}$	$3nT_{eG} + nT_{mG} + nsT_{mz} + nT_h$
Computational cost on MCSS/CSS	$2T_p + (c+1)T_{eG} + (c-1)T_{mG} + cT_{mZ} + T_s + (c+1)T_h$	$cT_{eG} + (c-1)T_{mG} + cT_{mZ}$	$nT_h + cT_{mZ} + cT_{eG} + cT_{mZ}$	$2csT_{mz} + 2cT_{eG} + (c-1)T_{mG}$
Computational cost on TPA	$T_p + T_{eG} + T_{mG_T} + cT_{mZ} + 2T_s + (2c+1)T_h$	$(c+2)T_p + T_{eG} + cT_{mG_T} + T_s + cT_{h2G}$	$(c+2)T_{h2G} + 2cT_{eG} + 2(c-1)T_{mG} + 2T_p + 2T_{mG}$	$2T_p + (c+s)T_{mG} + (2+s)T_{eG}$



(a) Comparison for tag generation (DO)

(b) Comparison for proof generation (MCSS/CSS)

(c) Comparison for verification (TPA)

(d) Comparison for batch auditing (MCSS/CSS and TPA)

Fig. 3. Experimental results of computational cost comparison.

in [16], [23], [24] when the challenged blocks change from 100 to 1000, and the growth rate of time overhead is much lower than that in the other protocols [16], [23], [24] as the number of blocks increases. More specifically, the time consumption of DO for tag generation of our protocol is just 30% and 24% of the time cost in Shen *et al.*'s protocol [16] and Hahn *et al.*'s protocol [24], respectively. In other words, compared with other related protocols, our protocol saves at least 2/3 of the computational cost on DO side. Fig. 3(b) shows the computational cost of MCSS/CSS for proof generation, from which we can see that the computational efficiency of our protocol is similar with others. The computational cost of TPA for proof verification is shown in Fig. 3(c), from which we can see that our protocol significantly reduces the computational cost of TPA. Specifically, the time consumption of our protocol is always less than 0.1 s since TPA does not need to perform any operations with high computational complexity, while the time consumption of Shen *et al.*'s protocol [16] and Hahn *et al.*'s protocol [24] increases linearly with the increase in the number of challenge blocks. Therefore, compared with other protocols [16], [23], [24], the computational cost on TPA side of our protocol is negligible. Finally, since Guo *et al.*'s protocol [23] and Hahn *et al.*'s protocol [24] do not support batch auditing for multi files, so we just compare the computational efficiency of batch verification with that in Shen *et al.*'s protocol

[16]. The comparison result is shown in Fig. 3(d), in which we assume that TPA verifies multi files at once and each file is 16 kb with 10 blocks. Besides, we counted both the time consumption of proof generation and proof verification in the batch auditing situation. We can see from the figure that our protocol has a distinct advantage in computational efficiency of batch verification. In summary, compared with the protocols [16], [23], [24], our protocol is computationally efficient both in single file auditing and batch auditing for multi-files. It significantly improves the computational efficiency of DO and TPA, and the computational efficiency of MCSS does not decrease. Therefore, the proposed protocol is more computationally efficient for real-environment applications.

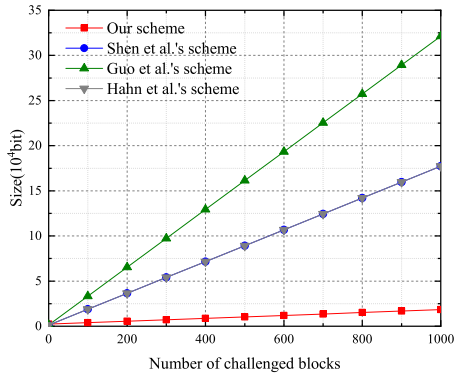
C. Communication Cost Comparison

In this section, we evaluate another aspect of the performance, i.e., the communication overhead, and compare our protocol with other related protocols [16], [23], [24]. We also show the communication overhead comparison in both table and figure forms. In the comparison, the communication overheads costed by that TPA submits the challenge to MCSS/CSS and MCSS/CSS responds the proof to TPA are counted. Since the type A pairings (a.properties) is used for the evaluation, p is 160 bits length, the curve is based on a base field of 512 bits, and the bit lengths of G_1, G_2 (or G) and G_T are all 1024 bits. Besides, $|d|$ and $|q|$ represent the bit lengths of the index number of challenged blocks and the signature, and they are 16 bits and 160 bits, respectively.

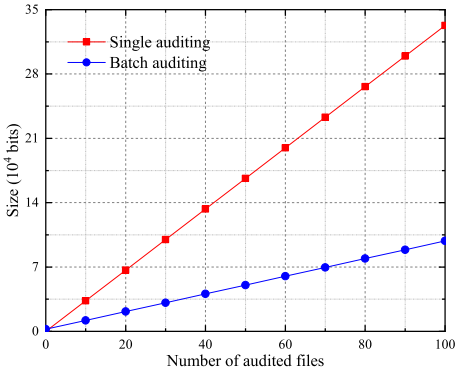
We count the communication overhead between MCSS/CSS and TPA for a challenge with c blocks. In our protocol, TPA sends a challenge $chal = \{IDX = \{j\}_{j\in[1,c]}, r\}$ to MCSS, and the communication overhead for this process is $c \cdot |d| + |p| = (160 + 16c)$ bits. In addition, MCSS responds a proof $\{PU, PV, SIG_{K_S}(PU, PV)\}$ to TPA, and the communication overhead is $2 \cdot |G_T| + |q| = 2208$ bits. In Shen *et al.*'s protocol [16], TPA challenges CSS and sends $chal = \{i, r_i\}_{i\in[1,c]}$ to CSS. The communication overhead of this process is $c \cdot |d| + c \cdot |p| = 176c$ bits. As a response, CSS submits a proof $\{T, D\}$ to TPA and the communication overhead is $|G| + |p| = 1184$ bits. In Hahn *et al.*'s protocol [24], the communication overhead between CSS and TPA is the same with that of Shen *et al.*'s protocol. In Guo *et al.*'s protocol [23], TPA challenges CSS and sends $chal = \{Q, K, B\}, Q = \{i_l, a_l\}_{l=1}^{c}, K \subset 1, \ldots, \lambda, |K| = 1$ and $B = \varnothing$ to CSS. The communication overhead of this process is $(c + 1) \cdot |d| + c \cdot |p| = 176c + 16$ bits where $|K| = 1$ is set as 16 bits. As a response, CSS submits a proof $\{\mu, \sigma, \sqcup_p\}$ to TPA and the communication overhead is $144c + 16 + |G| + |p| = 1200 + 144c$ bits supposing that each node cn has two children where cn presents the auxiliary information. Based on above analysis, we list the communication overhead comparison of three protocols for a single file challenge in Table V. It clearly shows that compared with other related protocols [16], [23], [24], our protocol significantly reduces the communication overhead between MCSS and TPA.

TABLE V
COMMUNICATION COST COMPARISON FOR A CHALLENGE

	Our protocol	[16]	[24]	[23]
TPA→MCSS	$c \cdot \|d\| + \|p\|$	$c{\cdot}\|d\|{+}c{\cdot}\|p\|$	$c{\cdot}\|d\|{+}c{\cdot}\|p\|$	$(c{+}1) \cdot \|d\| + c \cdot \|p\|$
bits	$= 160{+}16c$	$= 176c$	$= 176c$	$= 176c + 16$
MCSS→TPA	$2{\cdot}\|G_T\|{+}\|q\|$	$\|G\| + \|p\|$	$\|G\| + \|p\|$	$144c + 16 + \|G\| + \|p\|$
bits	$= 2208$	$= 1184$	$= 1184$	$= 1200 + 144c$



(a) Communication comparison of single file challenge



(b) Single auditing VS. batch auditing

Fig. 4. Communication cost comparison.

Fig. 4 shows a more intuitive comparison of the communication overheads. First, the communication comparison result of the different protocols in single file auditing is given in Fig. 4(a), where x-axis and y-axis represent the number of challenged blocks and the size of communication overhead, respectively. We can see from Fig. 4(a) that the growth rate of communication overhead in our protocol increases slowly as the number of challenged blocks increases, while the other three protocols [16], [23], [24] are just the opposite. Specifically, compared with other related protocols [16], [23], our protocol can save about 90% of communication overhead as the number of challenged blocks increases. Besides, we also give Fig. 4(b) to show the communication overhead comparison between single auditing and batch auditing of our protocol. In the comparison, we assume that each TPA challenges 60 blocks for each file. Besides, in the process of single auditing, TPA challenges the files one by one until the challenged files reach a certain number. In the Fig. 4(b), x-axis and y-axis represent the number of challenged

files and the size of the communication overhead, respectively, and it is clear that batch auditing is more efficient than single file auditing in communication. As the number of challenged files increases, the communication overhead of batch auditing is no more than 2% of the total communication overhead of single file auditing.

From the above analysis, we can conclude that our protocol performs better than other protocols [16], [23], [24] in terms of communication overhead, especially as the number of challenged files increases.

VII. Conclusion

To ensure the integrity of medical data for cloud-based medical storage systems, we have proposed an efficient privacy-preserving semi-trusted TPA-based public auditing protocol. Compared with other previously reported work, our protocol not only ensures storage correctness of MCSS, but also guarantees the privacy of DO's data against semi-trusted TPA. Besides, our protocol can also resist common attacks. In the performance aspect, the experiment results show that our protocol not only greatly reduces the computational costs of TPA and DO, but also significantly saves the communication overhead between TPA and MCSS. Specifically, compared with other related work, the computational cost of TPA in our protocol is negligible, and DO saves more than $2/3$ of computational cost, while the computational complexity of MCSS has not increased. On the communication overhead part, our protocol saves nearly 90% of communication overhead between MCSS and TPA as the number of challenged blocks increases. In short, our protocol is performance-friendly for real applications.

In the future, we will focus on blockchain-based decentralized remote data integrity auditing in cloud storage, and solve the problems of failure of single point, performance bottleneck, and user privacy leakage in TPA-based protocols.

References

[1] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: The digitization of the world from edge to core," *Seagate*, 2018. [Online]. Available: https://www. seagate. com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper. pdf

[2] K. Huang, X.-s. Zhang, Y. Mu, F. Rezaeibagha, and X. Du, "Bidirectional and malleable proof-of-ownership for large file in cloud storage," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2021.3054751.

[3] Y. Yang, Y. Chen, and F. Chen, "A compressive integrity auditing protocol for secure cloud storage," *IEEE/ACM Trans. Netw.*, vol. 29, no. 3, pp. 1197–1209, Jun. 2021.

[4] L. Zhou, A. Fu, Y. Mu, H. Wang, S. Yu, and Y. Sun, "Multicopy provable data possession scheme supporting data dynamics for cloud-based electronic medical record system," *Inf. Sci.*, vol. 545, pp. 254–276, 2021.

[5] I. Jayaraman and A. S. Panneerselvam, "A novel privacy preserving digital forensic readiness provable data possession technique for health care data in cloud," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 5, pp. 4911–4924, 2021.

[6] L. Nate, "The third party data breach problem," 2017. [Online]. Available: https://digitalguardian.com/blog/third-party-data-breach-problem

[7] L. Fouche, "The BDO and auscert 2018/19 cyber security survey: Response not just prevention," 2019. [Online]. Available: https://bdoaustralia.bdo.com.au/acton/attachment/18110/f-6eba696f-b266-4b04-a4a0-a4d2d025c351/1/-/-/-/-/1113_18_19-Cybersecurity-Report.pdf?sid=TV2:E7n5LYC0o

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

[9] 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.*, 2008, pp. 1–10.

[10] 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.

[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] 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.

[13] J. Ni, Y. Yu, Y. Mu, and Q. Xia, "On the security of an efficient dynamic auditing protocol in cloud storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 10, pp. 2760–2761, Oct. 2014.

[14] C. Wang, S. S. 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.

[15] H. Yan, J. Li, J. Han, and Y. Zhang, "A novel efficient remote data possession checking protocol in cloud storage," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 1, pp. 78–88, Jan. 2017.

[16] 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 Secur.*, vol. 12, no. 10, pp. 2402–2415, Oct. 2017.

[17] X. Li, S. Liu, and R. Lu, "Comments on 'A public auditing protocol with novel dynamic structure for cloud data'," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 2881–2883, 2020.

[18] D. He, N. Kumar, H. Wang, L. Wang, and K.-K. R. Choo, "Privacy-preserving certificateless provable data possession scheme for big data storage on cloud," *Appl. Math. Comput.*, vol. 314, pp. 31–43, 2017.

[19] X. Zhang, J. Zhao, L. Mu, Y. Tang, and C. Xu, "Identity-based proxy-oriented outsourcing with public auditing in cloud-based medical cyber-physical systems," *Pervasive Mobile Comput.*, vol. 56, pp. 18–28, 2019.

[20] T. Wu, G. Yang, Y. Mu, F. Guo, and R. H. Deng, "Privacy-preserving proof of storage for the pay-as-you-go business model," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 2, pp. 563–575, Mar./Apr. 2021.

[21] K. He, J. Chen, Q. Yuan, S. Ji, D. He, and R. Du, "Dynamic group-oriented provable data possession in the cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1394–1408, May/Jun. 2021.

[22] M. Thangavel and P. Varalakshmi, "Enabling ternary hash tree based integrity verification for secure cloud data storage," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 12, pp. 2351–2362, Dec. 2020.

[23] W. Guo *et al.*, "Dynamic proof of data possession and replication with tree sharing and batch verification in the cloud," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2020.3022812.

[24] C. Hahn, H. Kwon, D. Kim, and J. Hur, "Enabling fast public auditing and data dynamics in cloud services," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2020.3030947.

[25] E. Daniel and N. Vasanthi, "ES-DAS: An enhanced and secure dynamic auditing scheme for data storage in cloud environment," *J. Internet Technol.*, vol. 21, no. 1, pp. 173–182, 2020.

[26] Y. Ping, Y. Zhan, K. Lu, and B. Wang, "Public data integrity verification scheme for secure cloud storage," *Information*, vol. 11, no. 9, pp. 1–16, 2020.

[27] H. Tian *et al.*, "Dynamic-hash-table based public auditing for secure cloud storage," *IEEE Trans. Serv. Comput.*, vol. 10, no. 5, pp. 701–714, Sep./Oct. 2017.

[28] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2008, pp. 90–107.