IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 24, 2019, accepted August 15, 2019, date of publication September 2, 2019, date of current version September 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2938976

Publicly-Verifiable Proofs of Storage Based on the Discrete Logarithm Problem

MIAOMIAO TIAN[1], SHIBEI YE[1], HONG ZHONG[1], FEI CHEN[2], CHUANG GAO[1], AND JIE CHEN[3,4]

[1]School of Computer Science and Technology, Anhui University, Hefei 230601, China
[2]College of Computer Science and Engineering, Shenzhen University, Shenzhen 518060, China
[3]Department of Computer Science and Technology, East China Normal University, Shanghai 200062, China
[4]Co-Innovation Center for Information Supply and Assurance Technology, Anhui University, Hefei 230601, China

Corresponding author: Miaomiao Tian (mtian@ahu.edu.cn)

This work was supported in part by the Special Foundation for Key Program of Science and Technology of Anhui Province under Grant 18030901027, in part by the National Natural Science Foundation of China under Grant 61872243, in part by the Open Foundation of Co-Innovation Center for Information Supply and Assurance Technology under Grant ADXXBZ201701, and in part by the Science and Technology Plan Projects of Shenzhen under Grant JCYJ20180305124126741.

ABSTRACT With the rapid development of cloud computing platforms, cloud storage services are becoming widespread in recent years. Based on these services, clients are able to store data on remote cloud servers and thereby saving their local storage. This greatly reduces the burden of clients, while it also brings certain security risks to the outsourced data. Among the risks, a critical one is data corruption, for example cloud servers may delete some rarely used outsourced data for cost saving. To prevent this risk, proof of storage (PoS) schemes are invented, which can validate the integrity of cloud data without downloading the entire data. The existing PoS schemes, however, mostly either involve complex operations e.g. bilinear pairings, or don't support public verifiability. To fill this gap, in this paper we construct a new PoS scheme that is publicly verifiable and only requires simple cryptographic computations. We prove that our scheme is secure under the discrete logarithm assumption, in the random oracle model. Furthermore, we also show how to extend the scheme to support data updates. Finally, we implement our scheme. The simulation results demonstrate that our scheme is more computationally-efficient than the publicly-verifiable PoS schemes of Shacham and Waters (Journal of Cryptology 2013).

INDEX TERMS Proof of storage, public verifiability, efficiency.

I. INTRODUCTION

Cloud storage service, like Amazon S3, is one of the main services provided by cloud computing platforms. As the data gathered by individuals and organizations is excessively growing, cloud storage service has become increasingly popular in the last few years. By this way, clients could outsource a large amount of data into cloud storage servers for reducing their local storage overhead, and meanwhile clients can also access their outsourced data anytime and anywhere. Cloud storage services benefit people in various contexts, however they also bring some grievous security risks [3]. For instance, the cloud storage providers may delete some rarely used outsourced data or conceal data corruption accidents for commercial reasons. Thus, checking the integrity of outsourced data is indispensable in cloud storage services.

The associate editor coordinating the review of this manuscript and approving it for publication was Vicente Alarcon-Aquino.

Proof of storage (PoS) refers to an effective solution of checking the integrity of outsourced data. Notably it doesn't require downloading the entire outsourced data. To do this, PoS schemes (e.g. [1], [2], [19], [20]) usually first split data into many blocks, calculate their tags, and then outsource the data together with all tags into the cloud. Because those block tags are homomorphic linear authenticators of data blocks and can be homomorphically aggregated into one authenticator, the cloud can prove the integrity of outsourced data in little cost. More specifically, to check the integrity of outsourced data, a verifier can issue random challenges to the cloud, then the cloud calculates and returns the corresponding proofs. If all proofs are valid, the verifier would believe that the outsourced data is still intact. Otherwise, it must be broken.

The first practical PoS schemes are respectively proposed by Ateniese *et al.* [1] and Juels and Kaliski [12]

in 2007; from then on, many new PoS schemes have been proposed, e.g. [2], [6], [7], [11], [19]–[25]. Generally speaking, PoS schemes can be classified into two categories: publicly-verifiable PoS and privately-verifiable PoS. In a privately-verifiable PoS scheme, a verifier could confirm the validity of the proofs generated by the cloud only when it knows a private key. Thus the verifier of a privately-verifiable PoS scheme must be data owner/client himself, otherwise when the private key is publicized, the cloud is able to forge valid proofs using the private key. In contrast, the verifier of a publicly-verifiable PoS scheme may be anyone since the proof verification process needs no secret. We argue that publicly-verifiable PoS schemes could prevent potential disputes between the cloud and clients, and are drawing increasing attention from research community (see [2], [6], [11], [19]–[24] for examples).

We however note that most of the existing publicly-verifiable PoS schemes either adopt some costly cryptographic tools such as bilinear pairings and map-to-point hash functions (introduced in [5]), or can only work over composite-order groups. To remove this problem, considerable efforts have been made by researchers. For example, Zhang *et al.* [25] recently have proposed several pairing-free PoS schemes based on the privately-verifiable PoS of [7]. Their schemes are efficient, while they are still privately verifiable since their verification procedures must take some secret information as input.

In this paper we propose a new pairing-free PoS scheme. Our scheme is publicly verifiable and more computationally efficient than prior publicly-verifiable ones because it just involves simple cryptographic operations over prime-order groups and ordinary hash functions (such as SHA-256). We prove that our scheme is secure in the random oracle model under the discrete logarithm (DL) assumption. To our knowledge, it's the first DL-based publicly-verifiable PoS scheme. (Notice that publicly-verifiable PoS schemes using parings like [19], [24] are based on stronger assumptions such as the computational Diffie-Hellman assumption.) Moreover, we also show how to extend the scheme to support data updates. Finally, we implement our scheme to evaluate its performance. The simulation results demonstrate that our scheme is more computationally-efficient than the publicly-verifiable PoS schemes of Shacham and Waters [20].

In summary, this work makes the following contributions:

- We propose a new publicly-verifiable PoS scheme that involves only simple computations over prime-order groups and ordinary hash functions. We prove that our scheme is secure in the random oracle model, assuming the DL assumption holds over some prime-order groups.
- We also demonstrate how to extend our scheme for handling fully dynamic data, i.e. we approbate modification, insertion and deletion operations on outsourced data.
- We conduct extensive experiments for evaluating the performance of our scheme. The simulation results also validate its efficiency.

The rest of this paper is organized as follows. Section II reviews related works. Section III introduces some preliminaries to be used in this work. The proposed scheme and its security proof are provided in Section IV. We show how to extend our scheme to support dynamic data in Section V. Section VI evaluates the performances of our scheme. Finally, we conclude this work in Section VII.

II. RELATED WORK

Ateniese *et al.* [1] and Juels and Kaliski [12] first independently introduce several practical PoS schemes (with different names), and suggest to use them to check the integrity of outsourced data. One main difference between their works is that Ateniese *et al.*'s schemes just ensure most of outsourced data is intact while Juels *et al.*'s schemes could ensure the entire outsourced data can be recovered by applying an erasure code to original data. Ateniese *et al.*'s schemes also support public verifiability as well as an unbounded number of verifications. Later, many other publicly-verifiable PoS schemes are proposed. For instance, Ateniese *et al.* [2] show how to construct publicly-verifiable PoS schemes from homomorphic identification protocols and also present a concrete scheme based on factoring. Chen *et al.* [6] recently also present some publicly-verifiable PoS schemes based on the RSA assumption. Because factoring/RSA-based cryptographic schemes can only work over composite-order groups, they will yield bigger parameters (and hence more overhead) than schemes working over prime-order groups for the same security level.

The first publicly-verifiable PoS scheme working over prime-order groups is designed by Shacham and Waters [19] upon the BLS short signatures [5]. The authors prove that their scheme is secure in the random oracle model under the computational Diffie-Hellman (CDH) assumption (denoted as CDH-SW). In the full version [20], they also give an analogue of CDH-SW based on the RSA assumption (denoted as RSA-SW). Following the pioneering work of Shacham and Waters, other CDH-based publicly-verifiable PoS schemes have also been proposed such as [21]–[24]. Wang *et al.* [24] construct a PoS scheme supporting data updates upon CDH-SW using Merkle hash tree (MHT) [15]. To protect data privacy, several privacy-preserving versions of CDH-SW have been presented e.g. [21]–[23].

One drawback in CDH-SW and its descendants is: they all utilize complicated bilinear pairings and map-to-point hash functions. This may restricts the scope of their applications. To overcome this restriction, Guan *et al.* [11] develop new publicly-verifiable PoS schemes upon indistinguishability obfuscation [9]. The computation overhead of their schemes is small for producing block tags and proofs, whereas it's unaffordable for verifying proofs. Because their verification procedures rely on indistinguishability obfuscation that is built from very expensive multilinear maps [8]. Zhang *et al.* [25] recently have proposed certain efficient PoS schemes upon the privately-verifiable PoS of [7]. However, their schemes cannot support public verifiability.

In this work, we aim to advance the area of PoS by proposing publicly-verifiable, more efficient, yet provably secure solutions for PoS. In the remainder of the paper, we focus on publicly-verifiable PoS and we may abbreviate it as PoS if no confusion arises.

III. PRELIMINARIES

In this section, we give some preliminaries, including system model, formal definitions of PoS and cryptographic background used in this work.

A. SYSTEM MODEL

As illustrated in Figure 1, our PoS system comprises of three entities: a cloud, a user, and an auditor. The cloud is a provider of the cloud storage services who owns significant storage space and can provide cheap storage services to clients. The user represents a client who produces a large amount of data and wants to outsource its data into the cloud. To check whether the outsourced data has been destroyed/damaged, the auditor comes on stage, who models any verifier (including the user himself). On behalf of the user, the auditor sends a random audit query (also referred to as a challenge) to the cloud. After receiving the challenge, the cloud generates a proof and then forwards it to the auditor. Finally, the auditor can check the integrity of the outsourced data by verifying the validity of the proof. If the proof is valid, then the outsourced data may remain intact. Otherwise, it must be broken.

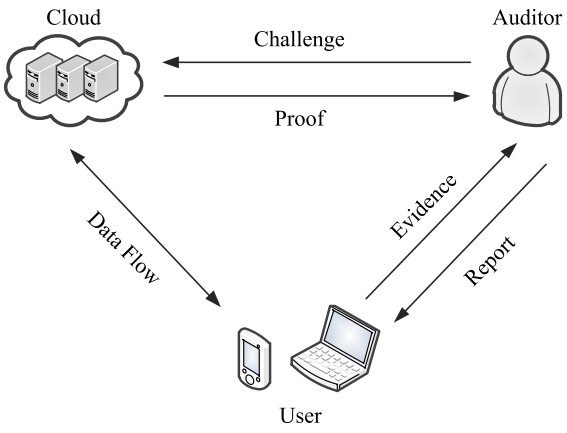


FIGURE 1. System model.

Similar to other PoS systems, in our system we always assume the user is fully honest and the cloud is malicious in the sense that it may conceal data losses.

B. PROOF OF STORAGE

We first introduce the syntax of PoS schemes.

Definition 1 (PoS): A PoS scheme consists of five probabilistic polynomial time (PPT) algorithms (`Setup`, `TagGen`, `Audit`, `Prove`, `Verify`), described as below:

- $\texttt{Setup}(\lambda) \rightarrow (PP, SK)$. This algorithm is run by the user. It takes as input a security parameter λ and outputs the system public parameters PP and a secret key SK.
- $\texttt{TagGen}(PP, SK, F) \rightarrow (T, E)$. This algorithm is also run by the user. It takes as input the system public parameters PP, the user's secret key SK, and a file F. It outputs a file tag T and an evidence E for T. The user then sends the file tag T and the evidence E to the cloud, and at the same time sends E to the auditor.
- $\texttt{Audit}(PP, E) \rightarrow \Lambda$. This algorithm is run by the auditor. It takes as input the system public parameters PP and an evidence E. It outputs a random audit request Λ. The auditor then sends Λ to the cloud.
- $\texttt{Prove}(PP, \Lambda, F, T) \rightarrow P$. This algorithm is run by the cloud. It takes as input the system public parameters PP, an audit request Λ, a file F and the corresponding file tag T. It outputs a proof P. The cloud then sends the proof P to the auditor.
- $\texttt{Verify}(PP, \Lambda, P, E) \rightarrow \{0, 1\}$. This algorithm is run by the auditor. It takes as input the system public parameters PP, an audit request Λ, a proof P, and an evidence E. It outputs 0 or 1. Output 0 indicates the proof is invalid; otherwise the proof is valid.

A PoS scheme must satisfy the following requirements:

- Correctness. If the data stored in the cloud is indeed intact and all entities are honestly follow the scheme, then the auditor must accept all proofs generated by the cloud.
- Security. If the auditor accepts the proofs for any outsourced data with non-negligible probability, then there exists a polynomial-time extraction algorithm that can recover a large fraction of original outsourced data (say 95%), except with negligible probability.

Formally, we have two definitions below.

Definition 2 (Correctness): We say a PoS scheme satisfies correctness if for all security parameter λ and data file F, let $(PP, SK) = \texttt{Setup}(\lambda)$, $(T, E) = \texttt{TagGen}(PP, SK, F)$ and $\Lambda = \texttt{Audit}(PP, E)$, if $P = \texttt{Prove}(PP, \Lambda, F, T)$, then the algorithm $\texttt{Verify}(PP, \Lambda, P, E)$ will output 1 with all but negligible probability.

For security, we consider the following game $\mathsf{Game}(\mathcal{C}, \mathcal{A})$ played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

- **Setup**. The challenger $\mathcal{C}$ runs the algorithm $\texttt{Setup}(\lambda)$ to obtain the system public parameters PP and the secret key SK. It then sends PP to the adversary $\mathcal{A}$.
- **Query**. The adversary $\mathcal{A}$ could issue TagGen queries to $\mathcal{C}$ adaptively. When $\mathcal{A}$ issues such a query on a file F, the challenger $\mathcal{C}$ obtains a file tag T and a corresponding evidence E by running the algorithm $\texttt{TagGen}(PP, SK, F)$ and then forwards them to $\mathcal{A}$.
- **Challenge**. When the above process ends, the challenger $\mathcal{C}$ issues an audit request Λ to the adversary $\mathcal{A}$ for checking the integrity of an outsourced file F. After receiving Λ, the adversary $\mathcal{A}$ outputs a proof P to $\mathcal{C}$.

Let the advantage $\mathsf{Adv}(\mathcal{A})$ of $\mathcal{A}$ with respect to a file F in the above game be the probability that the proofs generated by $\mathcal{A}$ for verifying file F are valid.

Definition 3 (Security): We say a PoS scheme is secure if for any PPT adversary $\mathcal{A}$ and any file F, the advantage $\mathsf{Adv}(\mathcal{A})$ of $\mathcal{A}$ with respect to the file F in $\mathsf{Game}(\mathcal{C}, \mathcal{A})$ is non-negligible, then there exists a polynomial-time extraction algorithm that can recover a large fraction of F with non-negligible probability.

C. CRYPTOGRAPHIC BACKGROUND

1) DISCRETE LOGARITHM PROBLEM

The security of our scheme relies on the hardness of the well-studied DL problem [16]. Roughly speaking, an instance of the DL problem is: given some fixed prime p, a cyclic group $\mathbb{G}$ of order p with generator $g \in \mathbb{G}$ and a random $u \in \mathbb{G}$, output $x \in \mathbb{Z}_p$ such that $u = g^x$. For a large prime p, this problem is conjectured to be intractable in general.

Definition 4 (DL Assumption): Given a large prime p, a cyclic group $\mathbb{G}$ of order p with generator $g \in \mathbb{G}$, and a random $u \in \mathbb{G}$, it's computationally infeasible to get $x \in \mathbb{Z}_p$ satisfying $u = g^x$.

2) DIGITAL SIGNATURE SCHEMES

Digital signature is a fundamental cryptographic primitive that can be used to achieve unforgeability of messages. A digital signature scheme consists of a signing algorithm $\mathtt{Sig}(\cdot)$ and a verification algorithm $\mathtt{Ver}(\cdot)$, where $\mathtt{Sig}(\cdot)$ takes a secret signing key and a message as input and outputs a signature, while $\mathtt{Ver}(\cdot)$ takes a public verification key, a message and its signature as input and outputs **accept** or **reject**. For any signing-verification key pair $(\mathsf{sk}, \mathsf{vk})$ and any message m, we require $\mathtt{Ver}(\mathsf{vk}, m, \mathtt{Sig}(\mathsf{sk}, m)) \rightarrow$ **accept**. We say a digital signature scheme is secure if it's difficult for any PPT adversary, who has obtained signatures $\mathtt{Sig}(\mathsf{sk}, m_i)$ of all messages m_i in a set $\mathcal{M}$, to output a signature $\mathtt{Sig}(\mathsf{sk}, m)$ of message m such that $m \notin \mathcal{M}$ and $\mathtt{Ver}(\mathsf{vk}, m, \mathtt{Sig}(\mathsf{sk}, m)) \rightarrow$ **accept**.

3) SECURE HASH FUNCTIONS

Hash functions are used extensively in cryptosystems. Usually, we say a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ is secure if it's collision resistant, i.e., it is intractable to find a pair $x, x' \in \{0,1\}^*$ such that $x \neq x'$ and $H(x) = H(x')$. In this paper, we only need ordinary hash functions such as SHA-256 that are very efficient in comparison to the map-to-point hash functions.

4) MERKLE HASH TREE

Merkle hash tree (MHT) is a well-studied authentication structure [15], in which only simple hash operations are involved. A MHT is usually constructed as a binary tree. Every non-leaf node in a MHT is a hash of its ordered child nodes and all leaf nodes are hashes of messages. Let $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ be a secure hash function such as SHA-256. Figure 2 illustrates an example of MHT. The messages are z_i $(i \in \{1,2,3,4\})$, and their hashes $h_i = H(z_i)$ are the leaf nodes of the tree. All non-leaf nodes are hashes of their two child nodes, e.g. the root node $h_0 = H(h_a, h_b)$. With the value of root node, a verifier can authenticate the validity of messages via auxiliary authentication information (AAI) with respect to their hashes. For instance, in order to authenticate z_2 of Figure 2, a prover returns to a verifier $\{z_2, \Omega_2\}$, where $\Omega_2 =< h_1, h_b >$ is the AAI of leaf node $h_2 = H(z_2)$. The verifier then calculates $h_2 = H(z_2), h_a = H(h_1, h_2)$ and checks whether $h_0 = H(h_a, h_b)$. If so, the verifier confirms the validity of z_2; otherwise, it's invalid.

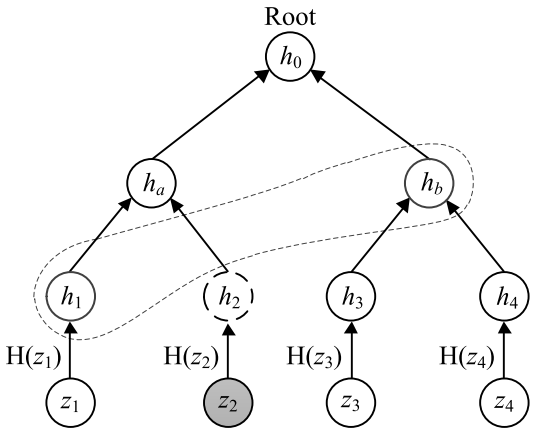


FIGURE 2. Merkle hash tree example.

For notation convenience, in this paper we treat the leaf nodes of a MHT as a left-to-right sequence; thus any leaf node can be uniquely determined by complying with this sequence.

IV. OUR SCHEME

In this section, we describe our DL-based PoS scheme. Throughout the rest of the paper, for a positive integer n, we let $[1, n] = \{1, \ldots, n\}$; $(\mathtt{Sig}(\cdot), \mathtt{Ver}(\cdot))$ indicates a secure DL-based signature scheme e.g. the Schnorr signature [18]; $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ is a secure hash function used in MHTs such as SHA-256; any outsourced file F will be divided into n blocks, each s sectors long. Let each sector be one element of $\mathbb{Z}_p$, then $F = \{m_{ij}\}_{i \in [1,n], j \in [1,s]}$, where $m_{ij} \in \mathbb{Z}_p$.

A. CONSTRUCTION

- $\mathtt{Setup}(\lambda)$: Given a security parameter λ, the user first chooses a prime $p = \Theta(2^\lambda)$, a cyclic group $\mathbb{G}$ of order p with generator $g \in \mathbb{G}$, a secure hash function $f : \{0,1\}^* \rightarrow \mathbb{Z}_p$, and a random signing-verification key pair $(\mathsf{sk}, \mathsf{vk})$. It then chooses random $y_1, y_2, \ldots, y_s$ from $\mathbb{Z}_p$ and calculates $Y_1 = g^{y_1}, Y_2 = g^{y_2}, \ldots, Y_s = g^{y_s}$. The system public parameters $PP = (p, \mathbb{G}, g, f, \mathsf{vk}, Y_1, Y_2, \ldots, Y_s)$ and the user's secret key $SK = (\mathsf{sk}, y_1, y_2, \ldots, y_s)$.
- $\mathtt{TagGen}(PP, SK, F)$: Given a file F, the user first divides it into $F = \{m_{ij}\}_{i \in [1,n], j \in [1,s]}$, where $m_{ij} \in \mathbb{Z}_p$. Then the user chooses a random file identifier id from $\mathbb{Z}_p$ and for each $i \in [1, n]$, calculates $\alpha_i = f(id, i) + \sum_{j=1}^{s} m_{ij} \cdot y_j$ and $\sigma_i = g^{\alpha_i}$. The block tag for m_i is σ_i and

the file tag is $T = (id, \sigma_1, \sigma_2, \ldots, \sigma_n)$. Next, the user generates the root R of a MHT, where the leaf nodes of the tree are the hashes of the ordered n index-tag pairs (from $(1, \sigma_1)$ to (n, σ_n)). The user using the above signature scheme signs (id, R) under the signing key sk and gets a signature $S = \mathtt{Sig}(\mathsf{sk}, id, R)$. The evidence of T is $E = (id, S)$. Finally, the user sends (F, T, S) to the cloud and E to the auditor, then deletes them from its local memory.

- $\mathtt{Audit}(PP, E)$: To check the integrity of the file F, the auditor chooses a random $I = \{(i, v_i)\}$ where $i \in [1, n]$ and $v_i \in \mathbb{Z}_p$, and then issues an audit request $\Lambda = (id, I)$ to the cloud.
- $\mathtt{Prove}(PP, \Lambda, F, T)$: After receiving $\Lambda = (id, I)$, the cloud sends a proof $P = (\mu_1, \ldots, \mu_s, \{(i, \sigma_i), \Omega_i\}_{i\in I})$ to the auditor, where $\mu_j = \sum_{i\in I} v_i \cdot m_{ij}$ for $j \in [1, s]$ and Ω_i is the AAI of $H(i, \sigma_i)$.
- $\mathtt{Verify}(PP, \Lambda, P, E)$: When the auditor has received the proof $P = (\mu_1, \ldots, \mu_s, \{(i, \sigma_i), \Omega_i\}_{i\in I})$ from the cloud, it generates the root R' using $\{(i, \sigma_i), \Omega_i\}_{i\in I}$ and verifies if $\mathtt{Ver}(\mathsf{vk}, S, (id, R')) \rightarrow \mathsf{accept}$. If so, it calculates $\nu = \sum_{i\in I} v_i \cdot f(id, i)$ and verifies whether

$$\prod_{i\in I} (\sigma_i)^{v_i} = g^{\nu} \cdot \prod_{j=1}^{s} Y_j^{\mu_j}$$

If so, the algorithm outputs 1; otherwise, it outputs 0.

Remark: We can observe that the algorithm $\mathtt{Verify}$ of this scheme does not need any secret information, so our scheme supports public verifiability.

Correctness. We can easily check that if all entities are honest, then the root generated using $\{(i, \sigma_i), \Omega_i\}_{i\in I}$ is always R, and hence $\mathtt{Ver}(\mathsf{vk}, S, (id, R))$ must output accept. Additionally, we have

$$\begin{aligned}
\prod_{i\in I} (\sigma_i)^{v_i} &= \prod_{i\in I} g^{\alpha_i \cdot v_i} \\
&= \prod_{i\in I} g^{f(id,i)\cdot v_i + \sum_{j=1}^{s} m_{ij}\cdot y_j \cdot v_i} \\
&= g^{\nu} \cdot \prod_{i\in I} \left(\prod_{j=1}^{s} Y_j^{m_{ij}\cdot v_i} \right) \\
&= g^{\nu} \cdot \prod_{j=1}^{s} Y_j^{\mu_j}
\end{aligned}$$

Therefore, the correctness follows.

B. SECURITY

Here we prove the security of our scheme. Our proof follows the proof strategy of Shacham and Waters [20], where a security proof consists of three parts. The first part shows that no PPT adversary can forge a proof for some challenge with non-negligible probability such that the proof is different from those computed as in real scheme but can still pass the auditor's verification. The second part shows that if one has received valid proofs for certain challenges, then there is a polynomial-time algorithm which is able to obtain a large fraction of data blocks of a file. Finally, the third part shows that these obtained blocks can be used to recover the full original file via an erasure code.

Note that in this work we only want to verify the integrity of a large fraction of outsourced data, so we don't use erasure codes in our scheme. We also remark that $(\mu_1, \mu_2, \ldots, \mu_s)$ in our scheme is the same as that in [20], thus the second part security proofs of [20] can be directly applied to ours. Therefore, in what follows we only need proving that all proofs passing the auditor's verifications are identical to the ones computed as in our real scheme, with all but negligible probability.

Theorem 1: If the discrete logarithm assumption holds in $\mathbb{G}$, then in the random oracle model no PPT adversary could provide valid proofs in $\mathsf{Game}(\mathcal{C}, \mathcal{A})$ with non-negligible probability such that the proofs are different from those computed as in our scheme.

Proof: Suppose that there exists a polynomial-time adversary $\mathcal{A}$ who could provide proofs that are different from those of our scheme but can pass the auditor's verifications with non-negligible probability, then we can construct an efficient algorithm $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine to solve the DL problem or to find a collision of H with non-negligible probability. Here H is modeled as a random oracle, and the input of the DL problem is a generator g of $\mathbb{G}$ and a random $u \in \mathbb{G}$. Algorithm $\mathcal{B}$ does so by interacting with $\mathcal{A}$ as follows.

Setup. Given a prime p, a generator g of $\mathbb{G}$ and a random element $u \in \mathbb{G}$, the algorithm $\mathcal{B}$ chooses secure hash function $f : \{0, 1\}^* \rightarrow \mathbb{Z}_p$, random signing-verification key pair $(\mathsf{sk}, \mathsf{vk})$ of the DL-based signature scheme $(\mathtt{Sig}(\cdot), \mathtt{Ver}(\cdot))$ and random $\alpha_i, \beta_i \in \mathbb{Z}_p$ for $i \in [1, s]$. Then $\mathcal{B}$ calculates $Y_i = g^{\alpha_i} u^{\beta_i}$ $(i \in [1, s])$. The system public parameters $PP = (p, \mathbb{G}, g, f, \mathsf{vk}, Y_1, Y_2, \ldots, Y_s)$.

Query. The adversary $\mathcal{A}$ issues TagGen queries adaptively. When $\mathcal{A}$ queries the file tag of file F, $\mathcal{B}$ first obtains $F = \{m_{ij}\}_{i\in[1,n], j\in[1,s]}$, then chooses a random identifier id from $\mathbb{Z}_p$ and calculates the i-th block tag $\sigma_i = g^{f(id,i)} \cdot \prod_{j=1}^{s} Y_j^{m_{ij}}$ for each $i \in [1, n]$. The file tag is $T = (id, \sigma_1, \sigma_2, \ldots, \sigma_n)$. Observe that the block tags are equal to those of our real scheme. Next, $\mathcal{B}$ generates the root R of a MHT, where the leaf nodes of the tree are the hashes of the ordered n index-tag pairs (from $(1, \sigma_1)$ to (n, σ_n)). Using the signature scheme, $\mathcal{B}$ signs (id, R) under the signing key sk and gets a signature $S = \mathtt{Sig}(\mathsf{sk}, id, R)$. Finally, $\mathcal{B}$ returns (F, T, S) to $\mathcal{A}$. Notice that in the simulation $\mathcal{B}$ will also store (F, T, S) locally, but it's unconscious for $\mathcal{A}$.

Challenge. The algorithm $\mathcal{B}$ chooses a random set $I = \{(i, v_i)\}$ where $i \in [1, n]$ and $v_i \in \mathbb{Z}_p$, and then issues an audit request $\Lambda = (id, I)$ to $\mathcal{A}$.

Forge. $\mathcal{A}$ with non-negligible probability outputs a valid proof $P' = (\mu'_1, \mu'_2, \ldots, \mu'_s, \{(i, \sigma'_i), \Omega'_i\}_{i\in I})$ for an audit request $\Lambda = (id, I = \{(i, v_i)\})$ such that P' is not identical

to the proof $P = (\mu_1, \ldots, \mu_s, \{(i, \sigma_i), \Omega_i\}_{i\in I})$ computed as in our scheme.

Since the proof P' is valid, then we know

$$\prod_{i\in I}(\sigma_i')^{v_i} = g^{\nu} \cdot \prod_{j=1}^{s} Y_j^{\mu_j'} \tag{1}$$

and $\mathtt{Ver}(\mathsf{vk}, S, (id, R')) \rightarrow \mathsf{accept}$, where R' is the root of a MHT generated using $\{(i, \sigma_i'), \Omega_i'\}_{i\in I}$ and $\nu = \sum_{i\in I} v_i \cdot f(id, i)$. Algorithm $\mathcal{B}$ could compute the root R of a MHT generated upon $\{(i, \sigma_i), \Omega_i\}_{i\in I}$ and $\mu_j = \sum_{i\in I} v_i \cdot m_{ij}$ for $j \in [1, s]$. It's clear that $\mathtt{Ver}(\mathsf{vk}, S, (id, R)) \rightarrow \mathsf{accept}$ and

$$\prod_{i\in I}(\sigma_i)^{v_i} = g^{\nu} \cdot \prod_{j=1}^{s} Y_j^{\mu_j} \tag{2}$$

As the signature scheme $(\mathtt{Sig}(\cdot), \mathtt{Ver}(\cdot))$ is secure under the DL assumption, then we know $R' = R$. Now we consider the following two possible cases.

Case 1: $((i, \sigma_i'), \Omega_i') = ((i, \sigma_i), \Omega_i)$ for all $i \in I$. In this case, we know that there exists at least one $k \in [1, s]$ such that $\mu_k' \neq \mu_k$ as per our presupposition. Then $\mathcal{B}$ divides the two equations 1 and 2, and gets

$$\prod_{j=1}^{s} Y_j^{\mu_j' - \mu_j} = 1.$$

Let $\Delta\mu_j = \mu_j' - \mu_j$, the above equation implies

$$u = g^{-\frac{\sum_{j=1}^{s} \alpha_j \Delta\mu_j}{\sum_{j=1}^{s} \beta_j \Delta\mu_j}}.$$

That is, $\mathcal{B}$ solves the DL problem, unless $\sum_{j=1}^{s} \beta_j \Delta\mu_j = 0$. However, we argue that $\Pr\left[\sum_{j=1}^{s} \beta_j \Delta\mu_j = 0\right] = 1/p$ since $\{\Delta\mu_j\}$ are not all zero, and $\{\beta_j\}$ are information-theoretically hidden from $\mathcal{A}$ by the result of [17]. Therefore, we know $\mathcal{B}$ can solve the DL problem with non-negligible probability.

Case 2: $((k, \sigma_k'), \Omega_k') \neq ((k, \sigma_k), \Omega_k)$ for some $k \in I$. In this case, we have $\sigma_k' \neq \sigma_k$ or $\Omega_k' \neq \Omega_k$. If $\Omega_k' \neq \Omega_k$, since the two MHTs have one same root, then we know for some pair of nodes $(h_{\mathsf{L}}, h_{\mathsf{R}})$ in one tree, there exists a different pair of nodes $(h_{\mathsf{L}}', h_{\mathsf{R}}')$ in the other tree such that $H(h_{\mathsf{L}}, h_{\mathsf{R}}) = H(h_{\mathsf{L}}', h_{\mathsf{R}}')$. Therefore, $\mathcal{B}$ finds a collision of H.

Else if $\sigma_k' \neq \sigma_k$, then $H(k, \sigma_k') \neq H(k, \sigma_k)$ (otherwise $\mathcal{B}$ also finds a collision of H). However, we note that this indicates $\Omega_k' \neq \Omega_k$. By the above analysis, we know that $\mathcal{B}$ can still find a collision of H.

In short, when there exists $k \in I$ such that $((k, \sigma_k'), \Omega_k') \neq ((k, \sigma_k), \Omega_k)$, then $\mathcal{B}$ can find a collision of H. However, this contradicts our assumption that H is a random oracle because such random oracles are collision resistant as per [13].

Finally, we can conclude that assuming the DL assumption holds in $\mathbb{G}$, then in the random oracle model no PPT adversary could provide valid proofs in $\mathsf{Game}(\mathcal{C}, \mathcal{A})$ with non-negligible probability such that they are different from those computed as in our scheme, as required. ■

V. HANDLING DYNAMIC DATA

Our scheme already supports static data. In this section, we extend our scheme to efficiently handle fully dynamic data. In other words, we permit the user to perform block modification, insertion and deletion operations on outsourced data. We use the notation `Mod` to denote modification, `Ins` to denote insertion, and `Del` to denote deletion respectively. In the following we assume the file $F = \{m_1, \ldots, m_n\}$ and its tag $T = (id, \sigma_1, \ldots, \sigma_n)$ have already been stored in the cloud. The evidence $E = (id, S)$ has also been sent to the cloud and the auditor, where $S = \mathtt{Sig}(\mathsf{sk}, id, R)$.

Like CDH-SW/RSA-SW [20], each block tag σ_i in our scheme is also relevant with the block index i, which may not be immutable when updating data. Wang *et al.* [24] overcome this issue by replacing each block index i with data block m_i and then employing a MHT to ensure the completeness of the new block tag. Clearly, applying this method to our scheme will make our scheme complicated. To simplify updating process, we keep the existing block tags and their indexes unchanged when updating data and introduce an extra index table *tab* to record the relationships between tag indexes and block indexes. An example of a 5-length index table is shown in Table 1. The index table *tab* is public for all entities while can only be maintained by the user himself. We claim that all entities must access to *tab* when they ask for the tag index of a block. Let the length of *tab* initially be n, the number of premier file blocks. When a file block is inserted, the user increases n by 1, and sets the tag index of the inserted block is n and updates all block indexes as usual. If the user wants to delete a file block, it just updates all block indexes as usual. For block modification, *tab* has no change. Below are the details of our data updating operations.

TABLE 1. Example of index table.

Tag index	1	2	5	3	4
Block index	1	2	3	4	5

A. DATA MODIFICATION

Suppose the user wants to modify block m_i into m_i'.

1) The user keeps *tab* invariable and computes the block tag σ_i' of m_i', and then issues the request $(\mathtt{Mod}, id, i, m_i', \sigma_i')$ to the cloud.
2) When the cloud receives the request, it updates the file and responds the user with $\{(i, \sigma_i), \Omega_i, S\}$, where Ω_i is the AAI of $H(i, \sigma_i)$. Then it replaces $H(i, \sigma_i)$ with $H(i, \sigma_i')$ and updates the MHT accordingly.
3) Upon receiving the response, the user first retrieves the previous root R using $\{(i, \sigma_i), \Omega_i\}$ and then verifies if $\mathtt{Ver}(\mathsf{vk}, S, (id, R)) \rightarrow \mathsf{accept}$. If so, the user generates the new root R' of the updated MHT using $H(i, \sigma_i')$ and Ω_i, produces a signature $S' = \mathtt{Sig}(\mathsf{sk}, id, R')$, and sends (id, S') to both the cloud and the auditor. Otherwise, the user outputs $\perp$.

4) If the message received from the user is $\perp$, the auditor aborts. Otherwise, it replaces the old evidence with (id, S').

Figure 3 shows an example of this procedure.

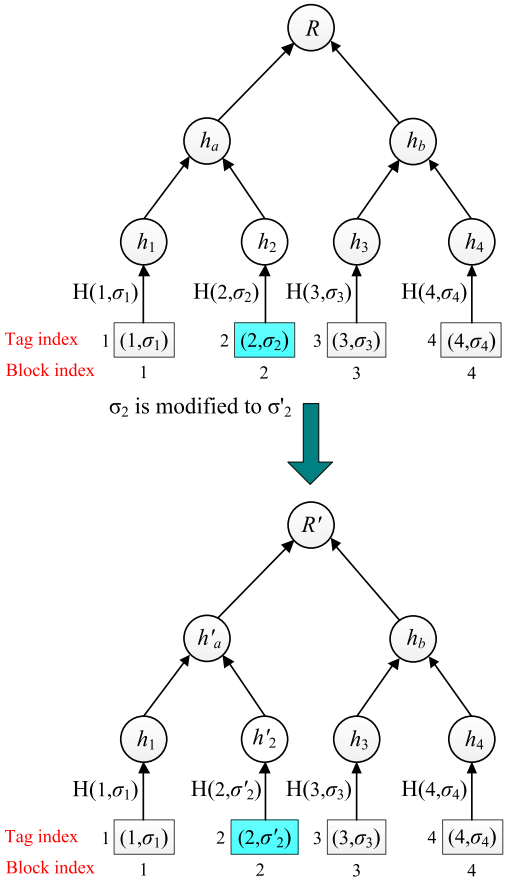


FIGURE 3. Example of block modification.

B. DATA INSERTION

Suppose the user wants to insert block m in file F after the i-th block m_i.

1) The user first increases the length n of *tab* by 1, sets the tag index of the inserted block is n and updates all block indexes. Then it publishes the updated index table *tab*, computes the block tag σ of m and issues the request $(\texttt{Ins}, id, i, m, \sigma)$ to the cloud.
2) After receiving the request, the cloud inserts m after m_i and responds the user with $\{(i, \sigma_i), \Omega_i, S\}$, where Ω_i is the AAI of $H(i, \sigma_i)$. Then it adds σ after σ_i and updates the MHT accordingly.
3) Upon receiving the response, the user first retrieves the previous root R using $\{(i, \sigma_i), \Omega_i\}$ and then verifies if $\texttt{Ver}(\mathsf{vk}, S, (id, R)) \rightarrow \textsf{accept}$. If so, the user replaces $H(i, \sigma_i)$ with $H(H(i, \sigma_i), H(i+1, \sigma))$, and generates the new root R' of the updated MHT using $H(i, \sigma_i)$ and Ω_i. Then it produces a signature $S' = \texttt{Sig}(\mathsf{sk}, id, R')$ and sends (id, S') to both the cloud and the auditor. Otherwise, the user outputs $\perp$.
4) If the message received from the user is $\perp$, the auditor aborts. Otherwise, it replaces the old evidence with (id, S').

Figure 4 shows an example of this procedure.

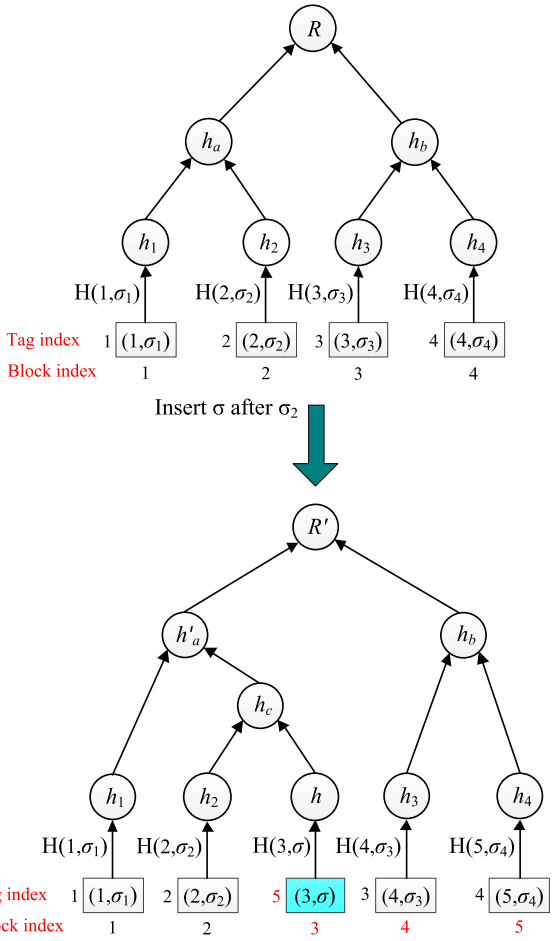


FIGURE 4. Example of block insertion.

C. DATA DELETION

Suppose the user wants to delete block m_i in the file F.

1) The user first updates all block indexes and publishes the updated index table *tab*. Then it issues the request $(\texttt{Del}, id, i)$ to the cloud.
2) After receiving the request, the cloud deletes m_i and responds the user with $\{(i, \sigma_i), \Omega_i, S\}$, where Ω_i is the AAI of $H(i, \sigma_i)$. Then the cloud deletes the leaf node $H(i, \sigma_i)$ and updates the MHT accordingly.
3) Upon receiving the response, the user first retrieves the previous root R using $\{(i, \sigma_i), \Omega_i\}$ and then verifies if $\texttt{Ver}(\mathsf{vk}, S, (id, R)) \rightarrow \textsf{accept}$. If so, the user generates the new root R' of the updated MHT using Ω_i, produces a signature $S' = \texttt{Sig}(\mathsf{sk}, id, R')$ and sends (id, S') to both the cloud and the auditor. Otherwise, the user outputs $\perp$.

4) If the message received from the user is $\perp$, the auditor aborts. Otherwise, it replaces the old evidence with (id, S').

Figure 5 shows an example of this procedure.

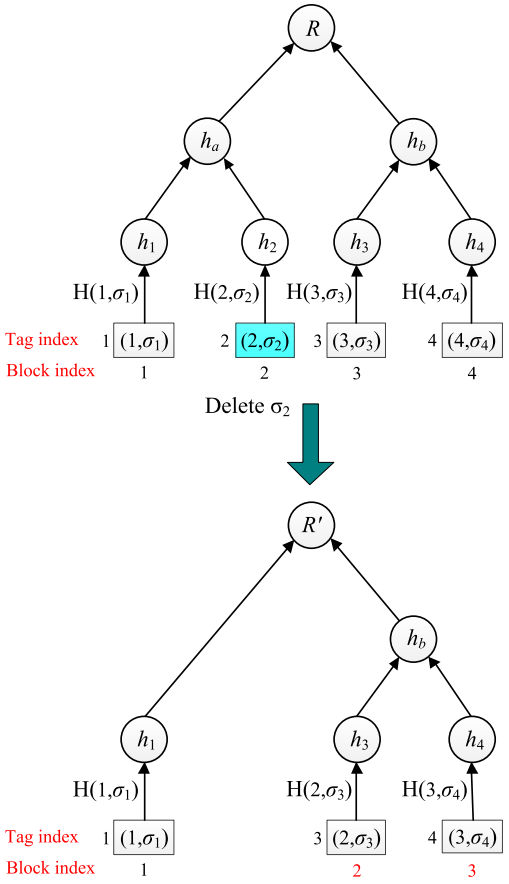


FIGURE 5. Example of block deletion.

VI. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our scheme. We will demonstrate the computation and communication costs of our scheme and also compare them with those of the publicly-verifiable schemes CDH-SW and RSA-SW in [20].

A. EXPERIMENT SETUP

The simulations are implemented in *C* language on a Ubuntu 18.10 system using Intel Core i5-4590 CPU at 3.30 GHz with 8 GB RAM. We set the security levels of all systems to be 128 bits. Hash functions are implemented using SHA-256. The implementation of CDH-SW utilizes the PBC library [14] and the Barreto-Naehrig curves [4] with base field size and group order length both are 256 bits. The messages in CDH-SW are signed by BLS short signature scheme [5]. We rely on the GMP library [10] to implement RSA-SW and our scheme, while employ RSA-PSS signature scheme of OpenSSL version 1.1.1 and Schnorr signature scheme [18] respectively to sign messages in RSA-SW and our scheme.

TABLE 2. Default parameters used in simulations.

Parameter	Default Value
File size	64 MB
Block size	4 KB
RSA modulus size	3072 bits
Number of challenged blocks	460

All simulation results represent the mean of 10 trials. Table 2 lists some default parameters used in simulations.

Before evaluating the performance of our scheme, we first determine the block size by analyzing the impact of various block sizes on the computation costs of CDH-SW, RSA-SW and our scheme. Specifically, we focus on the computation overhead of the user for generating file tags and the computation overhead of the auditor for verifying proofs, since both the user and the auditor have fewer computing resources than the cloud. Figure 6 illustrates the experimental result, in which we fix the file size to be 64 MB and the number of challenged blocks to be 460 for achieving a 99% confidence level [1].

The tag generation times of CDH-SW, RSA-SW and our scheme for various block sizes ranging from 2 KB to 32 KB are respectively shown in Figure 6(a), Figure 6(b) and Figure 6(c). Figure 6(d) depicts the impact of block size on the proof verification time of all three schemes. From these figures, we can find that the most balanced performance is achieved when the block size is 4 KB. Therefore, in what follows, we will fix the block size to 4 KB.

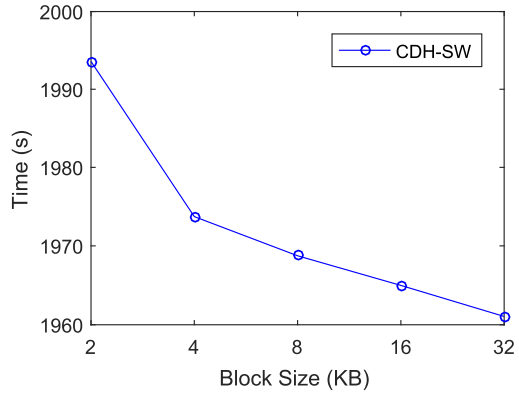
B. COMPUTATION COST

1) USER SIDE

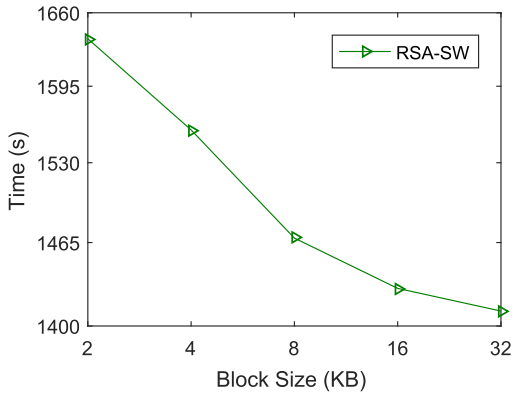
We first evaluate the computation overhead of the user. The most expensive operation for the user is calculating file tags and its computation time is determined by the size of files. In this simulation, we vary the sizes of files from 1 MB to 64 MB. The experimental result is shown in Figure 7. From the figure, we can see clearly that the computation time of the user in our scheme is much less than that in CDH-SW and RSA-SW. Specifically, this process in our scheme is about 250 times faster than that in CDH-SW and about 200 times faster than that in RSA-SW.

2) CLOUD SIDE

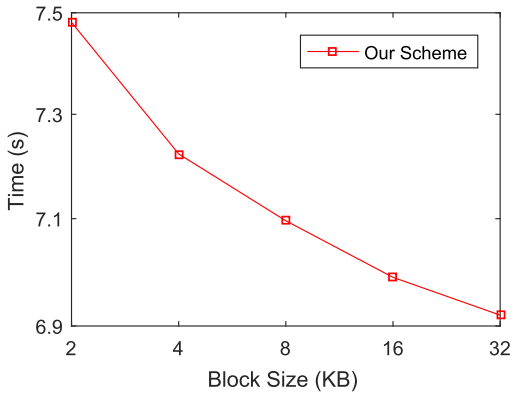
The computation overhead of the cloud is the cost of generating proofs for challenges from the auditor. In our scheme, each proof includes messages, tags and their corresponding AAIs. Thus, the computation cost of the cloud for one proof in our scheme is determined by the number c of indexes in I, i.e. the number of challenged file blocks, and the size of MHTs. In this simulation, we set the size of files to be 64 MB and vary the number c of challenged blocks from 100 to 500. The experimental result is illustrated in Figure 8(a). From the figure, we know that the computation cost of the cloud for one proof in our scheme is much less than that in CDH-SW



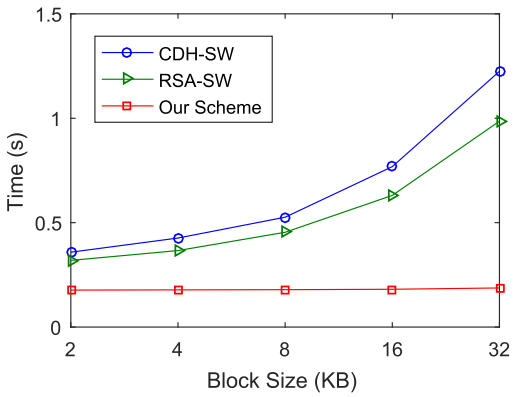
(a) Tag generation time for various block sizes in CDH-SW.



(b) Tag generation time for various block sizes in RSA-SW.



(c) Tag generation time for various block sizes in our scheme.



(d) Proof verification time comparison for various block sizes.

FIGURE 6. Computation overhead for various block sizes.

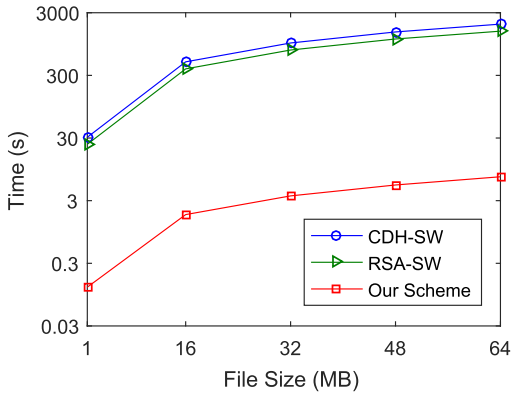


FIGURE 7. Tag generation time comparison for different file sizes.

and RSA-SW, and it grows very slowly with c. Moreover, the figure also indicates that the computation costs of the cloud in CDH-SW and RSA-SW increases quickly when c raises. This is due to the fact that the computations performed by the cloud in CDH-SW and RSA-SW are expensive.

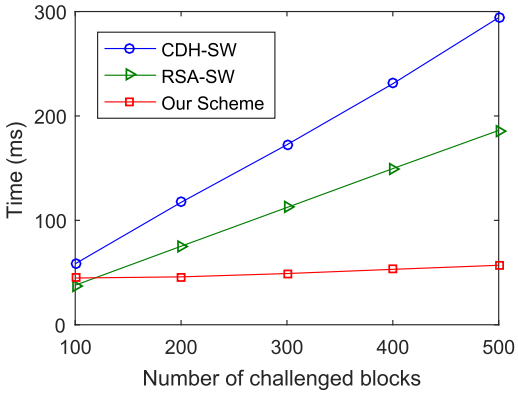
3) AUDITOR SIDE

The computation overhead of the auditor is dominated by the cost of verifying proofs. We show the computation time of the auditor when verifying one proof returned from the cloud in Figure 8(b). Here we still vary the number c of challenged blocks from 100 to 500. As the figure illustrates, the auditor in our scheme spends much less time to verify a proof. Specifically, the simulation result demonstrates that when $c = 460$ the proof verification process of our scheme is roughly 2.5 times faster than CDH-SW and 2.1 times faster than RSA-SW.

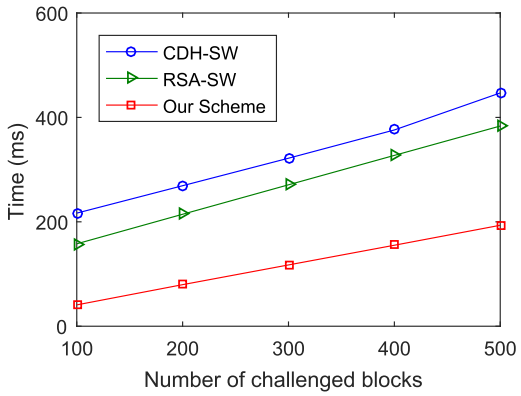
C. COMMUNICATION COST

Now we compare the communication overhead of our scheme with that of CDH-SW and RSA-SW. The communication overhead in these schemes is comprised of the user side communication overhead and the communication overhead between the auditor and the cloud. The user side communication overhead is introduced by user uploading files and their tags, while the communication overhead between the auditor and the cloud owes to issuing audit requests and returning relevant proofs. We compare the above two types of communication costs among CDH-SW, RSA-SW and our scheme in Figure 9(a) and Figure 9(b) respectively.

As the file size is identical for all schemes, in Figure 9(a) we just consider the communication overhead of uploading

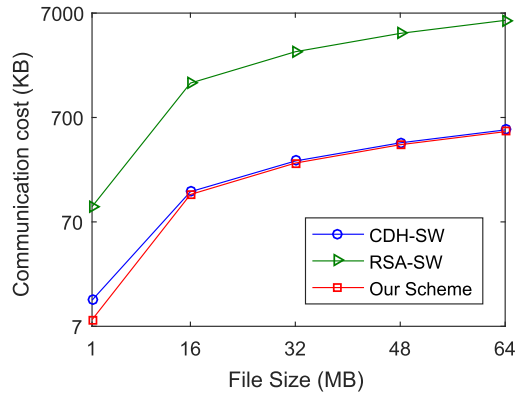


(a) Proof generation time comparison under different number of challenged blocks.

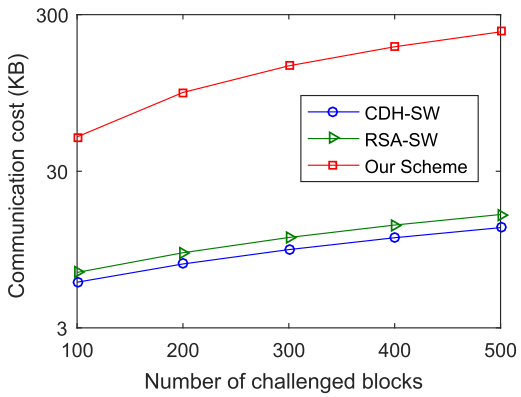


(b) Proof verification time comparison under different number of challenged blocks.

FIGURE 8. Computation overhead comparison for different number of challenged blocks.



(a) Communication overhead of the user for different file sizes.



(b) Communication overhead of one-round audit for different number of challenged blocks.

FIGURE 9. Communication overhead comparison.

file tags and range file sizes from 1 MB to 64 MB. From the figure, we can see that CDH-SW incurs a small additional overhead in communication on the user when compared to our scheme (here the block tags in CDH-SW are compressed). However, the user communication cost in our scheme is much less than that in RSA-SW. For the communication overhead between the auditor and the cloud, we challenge a fixed file of size 64 MB and range c from 100 to 500. Figure 9(b) shows that the costs of one-round communication between the auditor and the cloud in CDH-SW and RSA-SW are less than ours. (Yet, we argue that the communication overhead of our scheme can be greatly reduced if we let the auditor store the tags and the relevant AAIs received from the cloud. Then several times later the cloud does not have to send them in proofs any more.)

In summary, we know from the above simulation results that our scheme is much more efficient than both CDH-SW and RSA-SW in terms of computation overhead, while each of the three schemes has its own merit in terms of communication overhead.

VII. CONCLUSION

In this paper, we propose a practical publicly-verifiable PoS scheme based on the well-studied discrete logarithm problem and prove its security in the random oracle model. Our scheme removes the habitual bilinear pairings and map-to-point hash functions, and works well in prime-order groups. Furthermore, we also show how to extend the scheme to support dynamic data. Finally, we implement our scheme and the experimental results demonstrate its efficiency.

REFERENCES

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 598–609.

[2] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*. Berlin, Germany: Springer, 2009, pp. 319–333.

[3] Charles Babcock. (2014). *9 Worst Cloud Security Threats*. [Online]. Available: http://www.informationweek.com/cloud/infrastructure-as-a-service/9-worst-cloud-security-threats/d/d-id/1114085

[4] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proc. 12th Int. Workshop Sel. Areas Cryptogr. (SAC)*, Kingston, ON, Canada. Berlin, Germany: Springer-Verlag, Aug. 2005, pp. 319–331.

[5] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*. Berlin, Germany: Springer, 2001, pp. 514–532.
[6] F. Chen, T. Xiang, Y. Yang, and S. S. M. Chow, "Secure cloud storage meets with secure network coding," *IEEE Trans. Comput.*, vol. 65, no. 6, pp. 1936–1948, Jun. 2016.
[7] F. Chen, T. Xiang, Y. Yang, C. Wang, and S. Zhang, "Secure cloud storage hits distributed string equality checking: More efficient, conceptually simpler, and provably secure," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr./May 2015, pp. 2389–2397.
[8] S. Garg, C. Gentry, and S. Halevi, "Candidate multilinear maps from ideal lattices," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*. Berlin, Germany: Springer, 2013, pp. 1–17.
[9] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," *SIAM J. Comput.*, vol. 45, no. 3, pp. 882–929, 2016.
[10] T. Granlund and The GMP Development Team. (2016). *GNU MP: The GNU Multiple Precision Arithmetic Library, 6.1.2 Edition*. [Online]. Available: https://gmplib.org/
[11] C. Guan, K. Ren, F. Zhang, F. Kerschbaum, and J. Yu, "Symmetric-key based proofs of retrievability supporting public verification," in *Proc. Eur. Symp. Res. Comput. Secur. (ESORICS)*. Cham, Switzerland: Springer, 2015, pp. 203–223.
[12] A. Juels and B. Kaliski, "Pors: Proofs of retrievability for large files," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 584–597.
[13] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Boca Raton, FL, USA: CRC Press, 2014.
[14] (2007). *PBC Library*. [Online]. Available: https://crypto.stanford.edu/pbc/
[15] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. IEEE Symp. Secur. Privacy*, Apr. 1980, p. 122.
[16] A. Odlyzko, "Discrete logarithms: The past and the future," *Des. Codes Cryptogr.*, vol. 19, nos. 2–3, pp. 129–145, 2000.
[17] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proc. Annu. Int. Cryptol. Conf.* Berlin, Germany: Springer, 1991, pp. 129–140.
[18] C. P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, vol. 4, no. 3, pp. 161–174, 1991.
[19] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, vol. 5350. Berlin, Germany: Springer, 2008, pp. 90–107.
[20] H. Shacham and B. Waters, "Compact proofs of retrievability," *J. Cryptol.*, vol. 26, no. 3, pp. 442–483, Jul. 2013.
[21] M. Tian, L. Wang, H. Zhong, and J. Chen, "Attribute-based data integrity checking for cloud storage," *Fundam. Inform.*, vol. 163, no. 4, pp. 395–411, 2018.
[22] M. Tian, S. Ye, H. Zhong, L. Wang, F. Chen, and J. Cui, "Identity-based proofs of storage with enhanced privacy," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.* Cham, Switzerland: Springer, 2018, pp. 461–480.
[23] 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.
[24] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proc. Eur. Symp. Res. Comput. Secur. (ESORICS)*, 2009, pp. 355–370.
[25] J. Zhang, Y. Yang, Y. Chen, and F. Chen, "A secure cloud storage system based on discrete logarithm problem," in *Proc. IEEE/ACM Int. Symp. Qual. Service (IWQoS)*, Jun. 2017, pp. 1–10.

MIAOMIAO TIAN received the Ph.D. degree from the University of Science and Technology of China, in 2014. He is currently an Associate Professor with the School of Computer Science and Technology, Anhui University, China. His research interests include cryptography and information security.

SHIBEI YE is currently pursuing the master's degree with the School of Computer Science and Technology, Anhui University, China. Her research interests include cryptography and information security.

HONG ZHONG received the Ph.D. degree from the University of Science and Technology of China, in 2005. She is currently a Professor and the Dean of the School of Computer Science and Technology, Anhui University, China. Her research interests include cryptography, the IoT security, vehicular ad hoc networks, and software-defined networking.

FEI CHEN received the Ph.D. degree from the Chinese University of Hong Kong, in 2014. He is currently an Associate Professor with the College of Computer Science and Engineering, Shenzhen University, China. His research interests include information security and data privacy protection.

CHUANG GAO is currently pursuing the master's degree with the School of Computer Science and Technology, Anhui University, China. His research interests include cryptography and information security.

JIE CHEN received the B.S. degree from Soochow University, China, in 2008, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2013, both in mathematics. He was a Researcher with the ENS de Lyon, France, in 2016. He is currently a Professor with East China Normal University, China. His research interests include cryptography and information security.