PPTPS: Building privacy-preserving auditable service with traceable timeliness for public cloud storage

Li Li, Xiao Lan*, Mali Chen, Ting Luo, Li Chen, Yangxin Wang, Yumeng Chen

1 School of Artificial Intelligence, Chongqing University of Education, Chongqing, China, 2 Cybersecurity Research Institute, Sichuan University, Chengdu, Sichuan, China, 3 School of Mathematics and Big Data, Chongqing University of Education, Chongqing, China

* lanxiao@s cu.edu.cn

Abstract

Many works are designed to improve efficiency or enhance security and privacy of publicly-auditable cloud storage. However, building timeliness for cloud storage has not been well studied. Few works presented time-sensitive cloud storage and only focused on specific issues, such as the earliest creation time of files or resistance against a procrastinating auditor. Therefore, there leaves an absence of building traceable timeliness for publicly-auditable cloud storage. In this paper, we propose a solution PPTPS to build Privacy-Preserving auditable service with Traceable timeliness for Public cloud Storage. First, we use the security properties of the blockchain to provide a time-stamp for each phase. It enables the timeliness of cloud storage. Second, we construct efficient publicly-verifiable cloud storage. Third, a customized random mask solution is proposed to prevent privacy leakage during the auditing phase. Moreover, we formally proved the security of PPTPS. At last, experimental results demonstrate that PPTPS is economically sound and technically viable.

1 Introduction

Cloud storage service offers customers an efficient and economical way to store data and work together [1–3]. Many companies prefer to outsource data to cloud storage service providers, e.g., Dropbox, Amazon, and TortoiseSVN. However, data on the cloud often risks accidental loss or corruption from casual mistakes or hackers’ attacks [4]. Therefore, researchers have proposed publicly auditable solutions focusing on data integrity verification to address the data corruption problem in the past few years. Moreover, the cloud service providers need to preserve the outsourced data stably for archiving and keep track of the critical activities, such as outsourcing, auditing, proving, and verifying activities. Therefore, the traceable timeliness of data on the cloud is essential except for the integrity audit. For example, for effectively controlling COVID-19, tracing the earliest creation time of an infected patient’s data is critical. Also, in the online patent application, the timeliness of the patent determines the fairness of conclusions in judgments and dispute resolutions. Nevertheless, traditionally time-stamping schemes relied on a central service provider. Once this provider is compromised, attackers can
arbitrarily modify time stamps. Hence, a time-stamping solution is desirable to be free from the trust of the third time-stamping provider. Therefore, it is urgent to build traceable management for publicly-auditable cloud storage.

Overall, there are two limitations of previous works as follows. The first limitation is the absence of traceable timeliness for data on the cloud. However, few solutions are proposed to address the specific time-sensitive issues of cloud storage, e.g., resisting procrastinating auditors or determining the earliest data creation time. Moreover, the method to build the timeliness to provide traceable management for data on cloud storage is not well investigated. The second limitation is the existing publicly-auditable storage systems are not so efficient and have an information leakage problem during auditing. Lightweight publicly-auditable cloud storage with privacy-preserving enhancement is favorable in practice.

The first challenge is how to build the timeliness for cloud storage. The most popular solution is to make a time-stamp for each phase. Nevertheless, traditional solutions with trusted time-stamp service providers [5, 6] suffered from a single point of failure problem. Cao et al. utilized blockchain hashes to enable the timeliness of EHRs [7]. However, it cannot provide sufficient security by only using the latest block hash value due to the possibility of a blockchain fork. To cope with this problem, we adopt a fixed number of successive block hashes with the current block height [8], which are integrated into a block to provide an accurate time-stamp for each phase. Besides, these successive block hashes, the current block height, and other important information are recorded as entries for traceability on the cloud.

The second challenge is designing lightweight, privacy-preserving cloud storage without sacrificing security. To achieve more efficiency, we customize publicly-verifiable cloud storage through lightweight cryptographic tools and employ the random mask technique to enhance the privacy-preserving property during the audit phase.

In what follows, this paper achieves five design goals.

- **Public audit.** In contrast to the private audit, the public audit enables any party to check the data integrity on the cloud for various applications.
- **Traceability.** The timeliness is exhibited explicitly. We can get an accurate time-stamp of data in each phase. We can check the operations through the time-stamp of log entries stored on the cloud.
- **Auditing correctness.** Any data with corruption can pass the verification of an auditor with negligible probability.
- **Privacy-preserving.** Data content cannot be leaked to the auditor during the auditing phase. The security and privacy of data should be well protected.
- **Efficiency.** The computation and communication costs are acceptable in practice. Lightweight cryptographic tools are adopted.

This paper makes four main contributions as follows.

1. We propose a method to build traceable management for data on the cloud. For each phase of cloud storage, the corresponding information, the current block height, and a fixed number of successive block hashes are stored on the cloud and integrated into blockchain transactions. The timeliness based on the blockchain’s security properties explicitly provides us with the operation time of data for traceability.

2. We design a lightweight privacy-preserving publicly-auditable cloud storage system. We employ lightweight cryptographic tools rather than heavy ones to construct this system based on the DL assumption. Additionally, we enhance the privacy-preserving property of
the auditing process with a customized random mask method and ensure that the auditor cannot obtain any data from the information.

3. We formally prove security of PPTPS and further give the security analysis in four distinct aspects. In addition, we conduct extensive experiments to demonstrate PPTPS’s performance superiority to other works regarding computation and communication costs.

The construction of our article is organized as follows. Section 2 reviews the related works and Section 5 gives the problem statement. Background knowledge is included in Section 3 as preliminaries. Section 6 describes our scheme in detail and Section 7 gives the security analysis. Experimental results are demonstrated in Section 8. Finally, Section 9 concludes our article.

2 Related work

2.1 Secure auditable cloud storage

Since Ateniese et al. presented Proof of Data Possession (PDP) [9] and Juels et al. proposed Proof Of Retrievability (POR) [10], many works have emerged in the field of cloud storage [11–16]. First, Shacham et al. presented a solution SWP and adopted BLS signatures to support verifiability in [11]. Later, Wang et al. proposed a privacy-preserving publicly-auditable cloud storage system called PP-SWP by leveraging the random masking technique [12] to enhance the privacy-preserving property. Later, Chen et al. constructed a network coding-based public cloud storage system [13]. Both these two works support public verifiability but suffer from efficiency. Zhang et al. also used the DL assumption to support personal verification in [16]. However, these schemes are incapable of traceability. Tian et al. did not use the computationally complicated bilinear pairings for greater efficiency and used the DL assumption to construct a publicly-auditable cloud storage system [17].

2.2 Privacy-preserving cloud storage

Generally, a cloud storage system includes four phases [18–25]. First, the public and private keys for the cloud storage system are generated during the key generation phase. Next, data blocks with tags are outsourced to the CSP in the outsourcing phase. Moreover, an auditor generates a query and sends it to the CSP in the query phase. Then, in the proof generation phase, the CSP computes the corresponding proof and sends the proof to the auditor. Finally, the auditor validates the proof and checks whether the data on the CSP is intact or not in the verification phase. Moreover, to prevent the auditor from collecting the information during the auditing phase to derive the users’ data, some privacy-preserving techniques in distinct aspects are adopted, such as the random mask, the homomorphic secret sharing, and ring signature [12, 26, 27].

2.3 Blockchain-based retroactive storage

The blockchain enables medical centers to be built on different platforms to share EHRs without a third authority. In [7], a cloud-assisted eHealth system called CASES was proposed to resist illegal modifications of outsourced EHRs. In CASES, the auditor with a specific warrant can audit the outsourced EHRs. Hence, it did not support public verifiability well. Soon later, to construct a cloud storage system against procrastinating auditors, Zhang et al. proposed CPVPA, a public integrity verification method for cloud storage [8], by using the security properties of the blockchain. It enabled the detection of procrastinating auditors in time and allowed user checks. In [28], the nonces in a blockchain are used to construct unpredictable challenge data. Therefore, the forging auditing results from malicious TPAs can be prevented.
In [29], an accurate time stamping scheme, Chronos+, to judge the earliest creation time of a file was proposed by leveraging the blockchain’s chain growth property. However, these two works focused on specific storage problems, such as resisting procrastinating auditors and determining the earliest creation time of a file. Therefore, the solutions cannot support traceable management for storage straightforwardly. Later, Kim et al. presented a secure protocol for cloud-assisted EHR systems using the blockchain [30]. It comprised six phases: registration, authentication, smart contract uploading, EHR storing, EHR requesting, and log transaction uploading. However, these complicated phases brought a heavy burden regarding computation and communication costs. In a word, there is no traceable storage management concept in previous works. In [31], Xie et al., the authors applied smart contracts instead of untrusted third-party auditors to improve the reliability and stability of audit results. In [32], Zhang et al., the authors used the blockchain to record the interactions among users, service providers, and organizers in the data auditing process as evidence. Moreover, the smart contract was employed to detect service disputes.

For ease of explanation, the comparisons with previous works in different aspects are described in Table 1 as follows.

### Table 1. Comparisons with previous works.

| Works          | Traceability | All-phases management | Public audit | Privacy preserving in the audit phase |
|----------------|--------------|------------------------|--------------|---------------------------------------|
| SWP [11]       | x            | x                      | ✓            | x                                     |
| PP-SWP [12]    | x            | x                      | ✓            | x                                     |
| SCS-NC [13]    | x            | x                      | ✓            | x                                     |
| SCS-DLP [16]   | ✓            | x                      | x            | x                                     |
| CASES [7]      | ✓            | ✓                      | x            | ✓                                     |
| CPVPA [8]      | ✓            | x                      | ✓            | x                                     |
| Chronos+ [29]  | ✓            | x                      | ✓            | x                                     |
| CPIGS [38]     | x            | x                      | ✓            | ✓                                     |
| PPTPS          | ✓            | ✓                      | ✓            | ✓                                     |

In [29], an accurate time stamping scheme, Chronos+, to judge the earliest creation time of a file was proposed by leveraging the blockchain’s chain growth property. However, these two works focused on specific storage problems, such as resisting procrastinating auditors and determining the earliest creation time of a file. Therefore, the solutions cannot support traceable management for storage straightforwardly. Later, Kim et al. presented a secure protocol for cloud-assisted EHR systems using the blockchain [30]. It comprised six phases: registration, authentication, smart contract uploading, EHR storing, EHR requesting, and log transaction uploading. However, these complicated phases brought a heavy burden regarding computation and communication costs. In a word, there is no traceable storage management concept in previous works. In [31], Xie et al., the authors applied smart contracts instead of untrusted third-party auditors to improve the reliability and stability of audit results. In [32], Zhang et al., the authors used the blockchain to record the interactions among users, service providers, and organizers in the data auditing process as evidence. Moreover, the smart contract was employed to detect service disputes.

For ease of explanation, the comparisons with previous works in different aspects are described in Table 1 as follows.

### 3 Preliminaries

#### 3.1 Ethereum

The blockchain is a growing list of chained blocks. The blockchain is resistant to data modification. Once recorded into the blockchain, anyone cannot alter the data in the block retroactively without controlling more than the 51% hash rate. The blockchain includes two types: the public blockchain and the private blockchain. As one type of the most popular public blockchain, Ethereum is a network architecture composed of decentralized nodes called Ethereum nodes. Anyone with sufficient computer hardware can join the Ethereum network as a node and contribute computing power to earn block mining rewards. Up to 2022/09/17, about 10,490 Ethereum nodes were distributed worldwide.

Each block of Ethereum comprises two parts: header and body. The body includes the list of transactions. Moreover, the block header is more complex, containing the previous block’s hash, a time-stamp, mining difficulty, and other related parameters. Ethereum has two types of accounts: contract and externally owned accounts. A contract account has an ether balance, which stores the contract code that determines the ether change in the account.

The mining process is to pack the validated pending transactions into new blocks and use computing power to calculate the nonce value. The first miner who finds a nonce value and
broadcasts this value will be rewarded with a fee (Gas) deriving from the transactions within the block.

As a new block has been created, all nodes need to synchronize the block. Once all nodes accept the block, previously uncounted blocks will expire, and each node will recreate a block. The block-out time for each block is about 10s. As the computing power of the whole network keeps changing, the generation time of each block will be shortened as the computing power increases and lengthened as the computing power decreases [33].

On Ethereum, the blockchain height \( t \) denotes the current amount of blocks in the blockchain. Moreover, there are two properties of the blockchain as our fundamental to construct PPTPS [34–36].

- \( \phi \)-chain consistency: At any two rounds during an executing process, any two honest parties of the blockchain can differ only in the last \( \phi \) blocks. In Ethereum, the number of blocks is set as \( \phi \geq 12 \).

- Chain growth: During any time interval, the number of blocks chained into the blockchain is deterministic. Besides, in a given term, the block height increases steadily.

### 3.2 Bilinear maps

\( G_1 \) and \( G_2 \) are two multiplicative cyclic groups with large prime order \( p \). \( g_1 \) is the generator of \( G_1 \), and \( g_2 \) is the generator of \( G_2 \). \( e : G_1 \times G_2 \to G_T \) is the function that maps pairs of elements in \( G_1 \) and \( G_2 \) to elements in the cyclic group \( G_T \) with the prime order \( p \). The unique features of bilinear maps are listed as follows:

- Bilinear mapping: for \( \forall u \in G_1, \forall v \in G_2, \forall a, b \in \mathbb{Z}_p^* \), the equation \( e(u^a, v^b) = e(u, v)^{ab} \) holds.

- Non degenerate: \( \exists u, v \in G_1, \) such that \( e(u, v) \neq 1 \).

- Computational: for \( \forall u, v \in G_1, e(u, v) \) is computational in the polynomial time.

### 3.3 Complexity assumption

**Discrete logarithm problem.** In [37], the discrete logarithm problem is formally defined. Given a large prime \( p \), a cyclic group \( G \) with the order \( p \), a generator \( g \in G \), and a random value \( r \in G \), it is computationally infeasible to compute \( a \in \mathbb{Z}_p \) satisfying \( r = g^a \).

**The CDH assumption.** Given \( g, g^s, g_0 \in G_1 \), for unknown \( s \in \mathbb{Z}_q^* \), no probabilistic polynomial-time algorithm can compute \( g_0^s \) with non-negligible advantage.

### 4 Outline of publicly-auditable cloud storage

Generally, publicly-auditable cloud is composed of six algorithms including Setup, KeyGen, Outsource, Query, Prove, and Verify.

- **Setup:** With the security parameter, the data owner outputs the system parameters.

- **KeyGen:** This algorithm comprises two algorithms. In the algorithm SecretKeyGen the data owner outputs the private key \( sk \). In the algorithm PublicKeyGen the data owner outputs the public key \( pk \).

- **Outsource:** For each block, the data owner inputs the private key and generates a tag.
- **Query**: The auditor generates some random indexes with random values as an audit query and sends the audit query to the CSP.
- **Prove**: The CSP generates the proof for the query and returns the proof to the auditor.
- **Verify**: Upon receiving the proof, the auditor verifies the proof and outputs the result to demonstrate whether data on the CSP are intact or not.

## 5 Problem statement

First, we introduce the system model for the whole framework. Then we define the adversary model of PPTPS.

### 5.1 System model

As shown in Fig 1, six entities are included in PPTPS. The solid line represents the process, and the dotted line represents the message delivered.

- **The Cloud Server Provider (CSP)**: In PPTPS, CSP includes two parts. One part is a storage server that stores data and generates integrity proof traditionally. The other part is a log server. The log server is responsible for recording the corresponding time-stamp information.
- **The Data Owner**: The data owner is responsible for outsourcing the data to the CSP.
- **The Auditor**: The auditor generates an audit query and verifies the proof returned from the CSP.
• **Blockchain**: The blockchain provides the time-stamp functionality for PPTPS. In each phase, the blockchain outputs the current block height for other parties and then receives the transactions generated by other parties.

We describe Fig 1 briefly. First, the data owner outsources the data to the storage server of CSP. Subsequently, an auditor generates an audit query for the storage server. Then the storage server generates the corresponding proof. Later, the auditor verifies the proof to check whether the data on the storage server is intact or not. The data privacy is not leaked to the auditor during the audit phase. Besides, the log server of CSP records the log entries in each phase of PPTPS for traceability. The blockchain provides each phase’s hash values and the current block height. We build traceable management for a privacy-preserving publicly-auditable cloud storage system through the log entries and the blockchain.

5.2 The correctness definition

Here we present the correctness definition, which will be used in the subsequent sections.

**Definition 1 (Correctness)**: For the key $K$ and EHRs $F$, let $F = \text{Outsource}$, $q = \text{Query}$, and $\Gamma = \text{Prove}$, if the algorithm $\text{Verify}(\Gamma, K)$ outputs true with all but negligible probability, then PPTPS satisfies correctness.

5.3 Adversary model

In the adversary model, we categorize the threats into two types: internal and external adversaries. Furthermore, the internal adversary includes the malicious CSP, the procrastinating auditor, and the malicious data owner. The adversaries and their attack abilities are described as follows.

**The external adversary**: This adversary attempts to fore the tag of the block and breaks the security of a cloud storage system. Referring to [38], we present a generic framework for a game $\text{Game}_I(C, A_I)$, where a challenger is denoted as $C$ and an adversary is denoted as $A_I$. In $\text{Game}_I(C, A_I)$, there are three algorithms in the following.

- **Setup**: The challenger $C$ runs this algorithm to get the system parameters. Then the challenger $C$ sends the system parameters to the adversary $A_I$.

- **Queries**: The adversary $A$ adaptively generates different queries to the challenger $C$. Then the challenger $C$ responds to the queries to the adversary $A_I$.

- **Hash Query**: The adversary $A_I$ adaptively asks hash queries to the challenger $C$. $C$ responds the hash values to the adversary $A_I$.

- **Key Query**: The adversary $A_I$ adaptively asks key queries to the challenger $C$. $C$ executes the algorithm $\text{SecretKeyGen}$ and $\text{PublicKeyGen}$ to obtain the private key and the public key, respectively. $C$ returns the private key and the public key to $A_I$.

- **Tag Query**: The adversary $A_I$ adaptively chooses the block $d$ and sends $i$ to $C$ for querying the tag for this block.

- **Forge**: The adversary $A_I$ outputs a forged tag $s'$ for $d$.

  If the forged tag $s'$ for $d'$ is valid, then $A_I$ wins the game.

**Definition 2**: If $A_I$ wins $\text{Game}_I(C, A_I)$ with negligible probability, then the probability to forge a block tag is negligible.

From Definition 2, we obtain that without the correct private key, the adversary $A_I$ cannot forge the tag of its corresponding block.
The malicious CSP: Since data and tags are stored on the CSP, we must consider the problem that if the data is corrupted, the CSP will try to cheat the auditor that the data is still kept intact. We consider the malicious CSP as the adversary $A_{II}$. In the following GameII($C, A_{II}$), we focus on the problem that whether the adversary $A_{II}$ can forge a valid proof on corrupted data and pass the verification. In GameII($C, A_{II}$), there are four algorithms in the following.

- **Setup**: The challenger $C$ runs this algorithm to get the system parameters and the private key. Then the challenger $C$ keeps the private key and sends the system parameters to the adversary $A_{II}$.

- **SecretKey Query**: The adversary $A_{II}$ adaptively asks the secret key query to the challenger $C$. $C$ executes the **SecretKeyGen** algorithm to obtain the secret key and sends the secret key to $A_{II}$.

- **PublicKey Query**: The adversary $A_{II}$ adaptively asks the public key query to the challenger $C$. $C$ executes the **PublicKeyGen** algorithm to obtain the public key and sends the public key to $A_{II}$.

- **Tag Query**: The adversary $A_{II}$ adaptively chooses the block $d$ and sends $d$ to $C$ for querying the tag for this block. $C$ runs the algorithm **Outsource** and generates the tag for the block $d$. Finally, $C$ returns back the tag to $A_{II}$.

- **Forge**: $A_{II}$ forges the proof and sends the proof to $C$. If the proof can pass the verification without correct data, $A_{II}$ wins the GameII($C, A_{II}$).

**Definition 3**: If for the adversary $A_{II}$, $A_{II}$ wins the game GameII($C, A_{II}$) with negligible probability, then the probability to forge a valid proof without correct data is negligible.

**The procrastinating auditor**: The malicious auditors can perform the following attacks. First, a malicious auditor forges an entry that passes the verification successfully. Second, a malicious auditor forges an entry at the block height $t_1$ but claims that the entry is generated at the block height $t_2$, where $t_2 > t_1$.

### 6 Our scheme

#### 6.1 Construction of PPTPS

- **Setup**: With the security parameter $\kappa$, the following parameters are generated including a prime $p$ with bit length at least $\kappa$, two multiplicative cyclic groups $G_1$ (a generator for $G_1$ is $g$) and $G_2$ with the order $p$, and a secure hash function $H : \{0,1\}^* \rightarrow G_1^*$. The file $F$ with the name $fn$ is divided into $m$ blocks and each block is denoted as $d_i$, where $i = 1, 2, \ldots, m$. The system parameters for PPTPS are $(p, G_1, G_2, g, H)$. Besides, the data owner, CSP, and the auditor create externally owned accounts $A_D$, $A_O$, and $A_U$ in Ethereum, respectively.

- **KeyGen**: In the algorithm **SecretKeyGen**, the data owner randomly chooses $x_1, x_2, \ldots, x_m \in \mathbb{Z}_p^*$ as the private key $sk$. In the algorithm **PublicKeyGen**, the data owner computes $y_1 = g^{x_1}, y_2 = g^{x_2}, \ldots, y_m = g^{x_m}$ as the public key $pk$.

- **Outsource**: For each block $d_i$, using the private key $x_i$, the data owner computes a tag $s_i = g^d \cdot H(w)^h, w = \|fn\$. Then, the data owner extracts the block height $t_i$ and the block hash values of $\phi$ successive blocks $\{bl_{t_i-\phi+1}, bl_{t_i-\phi+2}, \ldots, bl_{t_i}\}$. Then the data owner outsources all the $(d_i, s_i)$ pairs and the values $\{bl_{t_i-\phi+1}, bl_{t_i-\phi+2}, \ldots, bl_{t_i}, t_i\}$ to the CSP. Upon receiving these values, the storage server checks $e(s_i, g)?(g^{d_i}, g)e(H(w), pk)$. If the
verification passes, the log server stores \(\{bl_{t_0}, bl_{t_0+1}, \ldots, bl_{t_1}\}\) and the storage server stores all the \((d, s)\) pairs. Otherwise, the storage server rejects the values. Then, the storage server computes \(\varphi = H(bl_{t_0}, \ldots, bl_{t_1})\). Then, the storage server generates a transaction \(T_2\) and sends it to the blockchain, where \(0\) ether is transferred from \(A_p\) to \(A_C\). The data value of the transaction \(T_2\) is set as \(\varphi\).

- **Query:** Randomized auditing is adopted to audit the data integrity. The auditor generates random indexes \((i_1, i_2, \ldots, i_m)\) with random values \((e_1, e_2, \ldots, e_m)\). Then the auditor extracts the current block height \(t_2\) and the block hash values \(\{bl_{t_2}, \ldots, bl_{t_2+i-1}\}\) from the blockchain and sends the values \(\{bl_{t_2+i}, \ldots, bl_{t_2+i+j-1}\}\) to the CSP. The auditor computes the hash value \(x = H(bl_{t_2+i}, \ldots, bl_{t_2+i+j-1})\). Subsequently, the auditor generates a transaction \(T_2\) to the blockchain with \(0\) ether transferring from \(A_U\) to \(A_C\). The data value of the transaction \(T_2\) is set as \(x\). At last, the auditor stores the entry \(\{bl_{t_2+i}, \ldots, bl_{t_2+i+j-1}\}\) in the log file on the log server.

- **Prove:** The customized random mask is used to enhance the privacy-preserving property. The storage server randomly chooses \(r \rightarrow Z_p\) and computes \(\theta = g^r, \omega = H(\theta)\).

\[
x' = \sum_{i=1}^{j} d_i e_{\alpha} = r \cdot \omega x, \text{ and } \beta = \theta \prod_{i=1}^{j} \xi_i.
\]

Afterward, the storage server returns back \(\Gamma = (\alpha, \beta, \theta)\) to the auditor as proof. Similarly, the storage server extracts the current block height \(t_4\) and the block hash values of \(s:\) successive blocks \(\{bl_{t_4}, \ldots, bl_{t_4+i-1}\}\) from the blockchain. The storage server computes the value \(\lambda = H(bl_{t_4}, \ldots, bl_{t_4+i-1})\) and sends a transaction \(T_3,\) where \(0\) ether transferring from \(A_C\) to \(A_U\) and the data value of the transaction \(T_3\) is set as \(\lambda\). Finally, the storage server stores the entry \(\{bl_{t_4+i}, \ldots, bl_{t_4+i+j-1}\}\) in the log file on the log server.

- **Verify:** Upon receiving the proof \(\Gamma = (\alpha, \beta, \theta)\), the auditor computes \(\omega = H(\theta)\) and checks whether \(e(\beta, g) = e(g^s, g) e(\prod_{j=1}^{i} H(w)^{\gamma_i}, pk)\) If yes, the auditor accepts the proof and outputs the result \(\xi\) with the value of \(True\). Otherwise, the auditor rejects the proof and outputs the result \(\xi\) with the value of \(False\). Later on, the auditor extracts the current block height \(t_4\) and the block hash values \(\{bl_{t_4+i}, \ldots, bl_{t_4+i+j-1}\}\) from the blockchain. The auditor computes \(\varphi = H(bl_{t_4+i}, \ldots, bl_{t_4+i+j-1})\) and sends a transaction \(T_4\), where \(0\) ether transferring from \(A_U\) to \(A_P\) and the data value of the transaction \(T_4\) is set as \(\varphi\). At last, the log server stores the log entry \(\{bl_{t_4+i}, \ldots, bl_{t_4+i+j-1}\}\) on the log server.

The phases' results are recorded in the blockchain transactions and stored in the log server of CSP. Therefore, any third party or user can check for authenticity by leveraging the blockchain and the stored log entries of the CSP.

### 6.2 Correctness analysis

In what follows, the correctness proof for PPTPS is given.

**Theorem 1:** If the corresponding entities follow under the steps in the phases of Delegate, Outsource, Query, Prove, and Verify strictly, the correctness of PPTPS can be demonstrated.

According to Definition 1, we can demonstrate the algorithm \(\text{Verify}(q, \Gamma, K)\) outputs the result of \(True\). We can demonstrate that the following Eq 1 holds.

\[
e(\beta, g) = e(g^s, g) e(\prod_{j=1}^{i} H(w)^{\gamma_i}, pk) \tag{1}
\]
7 Security analysis

We present a security analysis of PPTPS in three aspects. At first, we prove that the external adversary cannot forge the tag of the block through Theorem 2 and the malicious CSP cannot forge a valid proof on the corrupted data through Theorem 3. Then we give the security analysis for the procrastinating auditor and the malicious data owner, respectively. At last, we analyze the privacy-preserving property in the auditing query phase through Theorem 4.

Theorem 2: If the adversary \( A_x \) wins \( \text{Game}1(C, A_x) \) with the advantage \( \epsilon \), after making \( \text{SecretKey Query}, \text{PublicKey Query}, \text{Tag Query}, \text{Hash Query}, \text{Tag Query} \) at most \( q_{sk}, q_{pk}, q_{hp}, q_{t}, q_{T} \) times respectively, then a simulator \( B \) can break the CDH problem with the non-negligible probability.

Proof: Given a CDH instance \((g, G_1, g^e, g^k)\), if the adversary \( A_x \) wins \( \text{Game}1(C, A_x) \) with non-negligible probability, the simulator \( B \) can calculate \( g^{ab} \) at non-negligible probability by the capability of \( A_x \). \( B \) simulates each interaction step with \( A_x \) as follows.

- Setup: \( B \) generates the system parameters.

- SecretKey Query: \( A_x \) adaptively executes \( \text{SecretKey Query} \). Meanwhile, \( B \) maintains a list \( L_1 = (pk, sk, T) \).

  - If \( sk^e \) does not exist in \( L_1 \), \( B \) randomly chooses a value \( x^* \in Z_\alpha^\ast \) and flips a coin \( T^* \in \{0, 1\} \). Assume the probability of \( T^* = 0 \) is \( \gamma \) and the probability of \( T^* = 1 \) is \( 1 - \gamma \). If \( T^* = 1 \), \( B \) calculates \( pk = (g^e)^{x^*} \) and adds the tuple \( (pk^e, x^*, T^*) \) to \( L_1 \). Then \( B \) aborts and outputs ‘fail’. If \( T^* = 0 \), \( B \) calculates \( pk = g^x \) and adds the tuple \( (pk^e, x^*, T^*) \) to \( L_1 \). Then \( B \) returns \( x^* \) to \( A_x \).

  - If \( sk^e \) exists in \( L_1 \), \( B \) checks \( T^* \). If \( T^* = 1 \), \( B \) outputs ‘fail’ and abort. Otherwise, \( B \) retrieves \( sk^e \) and returns it to \( A_x \).

- PublicKey Query: \( A_x \) adaptively executes \( \text{PublicKey Query} \). Meanwhile, \( B \) maintains a list \( L_1 = (pk, sk, T) \).

  - If the tuple \( (pk^e, sk^e, T^*) \) already exists in \( L_1 \), \( B \) returns \( pk^e \) to \( A_x \) directly.
• If the list \( L_1 \) does not contain the tuple \((pk', sk', T')\), \( B \) randomly selects a value \( x' \in \mathbb{Z}_p^* \) and flips a coin \( T' \in \{0, 1\} \). Assume the probability of \( T' = 0 \) is \( \gamma \), and the probability of \( T' = 1 \) is \( 1 - \gamma \). For \( T' = 1 \), \( B \) calculates \( pk' = g^{x'} \). \( B \) adds a new tuple \((pk', x', T')\) into \( L_1 \) and returns back \( pk' \) to \( \mathcal{A}_I \).

• **Hash Query:** \( \mathcal{A}_I \) adaptively executes **Hash Query** for \( w \in \mathbb{Z}_p \). \( B \) also maintains a list \( L_2 = \{w, h\} \). If the list \( L_2 \) already contains \( w' \), \( B \) retrieves \( h' \) and returns \((g^h)^{h'} \) to \( \mathcal{A}_I \). Otherwise, \( B \) selects a random value \( h \in \mathbb{Z}_p \) and returns \((g^h)^h \) to \( \mathcal{A}_I \). Finally, \( B \) inserts \((w', h')\) into the list \( L_2 \).

• **Tag Query:** \( \mathcal{A}_I \) adaptively executes **Tag Query** with \((w', d')\). At first, \( B \) checks whether \( T' = 0 \) in the list \( L_1 \). If \( T' = 0 \), \( B \) gets \( sk' \) from \( L_1 \) and \( H(w') \) from \( L_2 \). Otherwise, \( B \) aborts and outputs 'fail'. Then \( B \) computes the tag for \((w', d')\) by the algorithm **Outsource** and returns the tag to \( \mathcal{A}_I \).

• **Forge:** \( \mathcal{A}_I \) outputs a tuple \( O = (s', w', d') \). \( s' \) is a forged tag on the block \( d' \).

• **Analysis:** If \( \mathcal{A}_I \) wins **GameI(C, A_I)**, \( B \) can get \( e(s', g) = e(g^{d'}, g)e(H(w'), pk') \). Then \( B \) retrieves the tuple \((pk', x', T')\) from \( L_1 \). If \( T' = 0 \), \( B \) aborts and outputs 'fail'. Otherwise, \( B \) gets \( pk' = g^{x'} \) from \( L_1 \) and \( H(w') = g^{\beta w} \) from \( L_2 \). Then \( B \) gets \( e(s', g) = e(g^{d'}, g)e(g^{\beta w}; g^{ax'}) \).

\[
\text{We get } g^{\alpha = (s')^\frac{1}{2\gamma}} \text{. Therefore, the probability that } B \text{ simulates the interactions with } \mathcal{A}_I \text{ without abortion is higher than } (1 - \gamma)^{\frac{1}{2\gamma}}. \text{ } B \text{ outputs } g^{\alpha} \text{ with the probability } e' \geq e(1 - \gamma)^{\frac{1}{2\gamma}} \geq \frac{\epsilon}{2^{(\alpha k + q)}}. \text{ Therefore, } B \text{ breaks the CDH problem with the non-negligible probability.}
\]

**Theorem 3:** If security for DLP is guaranteed, then the probability that the adversary \( \mathcal{A}_{II} \) wins **GameII(C, A_{II})** is negligible.

**Proof:** If \( \mathcal{A}_{II} \) outputs the integrity proof \( \Gamma = (\theta, \alpha, \beta) \) and wins **GameII(C, A_{II})** with non-negligible probability, we can get Eq 3.

\[
e(\beta_1, g) = e(g^{\alpha}, g)(\prod_{j=1}^{i}H(j||fn)^{\gamma_j}, pk).
\]

Because \( \mathcal{A}_{II} \) wins **GameII(C, A_{II})**, there exists \( \beta_1 = \beta \) but \( \alpha_1 \neq \alpha \). According Eqs 1 and 3, we define \( \Delta = \alpha_1 - \alpha \) and get \( g^{\Delta 2} = 1 \). Given \( h, u \in \mathbb{G}_1 \), we set \( u = h^x \) and compute \( x \in \mathbb{Z}_p \). Let \( g = h'^{u} \) where \( b \) is randomly chosen and \( a, b \in \mathbb{Z}_p^* \), we get \( (h^a u^b)^{\Delta 2} = 1 \) and \( u = h^{-a l b} \). Since \( b \in \mathbb{Z}_p \) is randomly chosen and information-theoretically hidden from the adversary [39], we get the right value \( x \) and solve DLP with a non-negligible probability \( 1 - 1/q \). To resist attacks from the internal adversaries in Section 5.3, we describe two security properties, respectively.

### 7.1 PPTPS guarantees the timeliness

In each phase, we integrate the corresponding information into a blockchain transaction. Hence, the block hashes and the block height in which the transaction is integrated reflect the timeliness of PPTPS. After the transaction is recorded, any third party can extract the corresponding block hashes and the block height. For example, in the **Verify** phase, the block height is \( t_s \); the user gets the log entry \([bl_{t_1 + \delta t_1}, bl_{t_2 + \delta t_2}, \ldots, bl_t, \theta, x, \beta]\) stored on the storage server and computes \( \lambda' = H(bl_{t_1 + \delta t_1} \ldots bl_t || \theta \| x \| \beta) \). Besides, the security of timeliness is guaranteed by the underlying blockchain. To our best knowledge, if an adversary is
without a 51% hash rate, it cannot break the security of the blockchain. Thus, timeliness security is well guaranteed.

7.2 Resistance against the procrastinating auditor

PPTPS can resist malicious auditors. First, since the security of PPTPS has been demonstrated in the game Game($C, A$), it is computationally infeasible for a malicious auditor to generate an entry {$bl_{t_{i-1}+1}, bl_{t_{i-1}+2}, \ldots, bl_{t_i}, t_i, \theta, x, \beta$} that passes the verification in the Verify phase. Second, Blockchain security guarantees timeliness. For the procrastinating auditor, it is computationally infeasible to generate a transaction at the block height $t_2$. Still, the malicious auditor convinces others that the transaction is generated at the block height $t_1$.

7.3 Privacy-preserving audit

The below Theorem 4 shows that the auditor cannot derive the data owner’s data from information generated during auditing.

**Theorem 4**: From the CSP’s response $\Gamma = (\theta, \alpha, \beta)$, the auditor cannot recover $\alpha'$.

**Proof.** Without random masking, the CSP computes $x = \sum_{j=1}^{\phi} d_{i,j}e_{i,j}, \beta = \prod_{j=1}^{\phi} s_{i,j}^{\phi}$ and sends $\Gamma = (\alpha, \beta)$ as the proof. However, in this way, there exists potential information leakage. Since by collecting a sufficient number of linear combinations of the same set of data blocks, the auditor can generate a group of linear equations and solve these equations to derive the data owner’s data. However, with random masking, the CSP computes $\theta = g^r, \omega = H(\theta), \alpha' = \sum_{j=1}^{\phi} d_{i,j}e_{i,j} r + \omega e^\alpha, \beta = \theta \prod_{j=1}^{\phi} s_{i,j}^{\phi}$ and sends $\Gamma = (\theta, \alpha, \beta)$ as the proof. Straightforwardly, due to the security properties of the random number and the hash function, the auditor cannot recover $\alpha'$. Therefore, the auditor cannot derive the data owner’s data.

8 Experimental evaluation

By conducting extensive experiments, we evaluate the performance of PPTPS in computation and communication costs. The experiments are performed on a Win7 operating system, an Intel Core 2 i5 CPU, and 8 GB DDR 3 RAM. Java Development Kit and Eclipse IDE for Java Developers are installed. We use java.security, java.math, and java.util packages to evaluate the computation and communication costs. Besides, all the EHRs datasets as benchmarks are public, which can be downloaded from [40]. The sizes of the selected five benchmarks range from 6KB to 13MB. The block size is set as 1024 bits, the number of successive blocks is set as $\phi = 12$, and the number of challenged blocks is set as 30 by default. We run ten times for each experiment and take the average value as the final result for more precision.

8.1 Computation cost

We observe the computation cost in each phase. In the Delegate and Query phase, the operations need few computation costs. Hence, we focus on the computation costs in the Outsource, Prove, and Verify phases. In Figs 2 and 3, it is observed that the computation costs in the phases of Prove and Verify vary slightly, within 1s with the increment of benchmark sizes. However, the computation cost in the Outsource phase increases dramatically with the increment of benchmark sizes.

In the following, we compare the computation cost with previous works. The basic cryptographic operations and their meanings of the computation cost are listed in Table 2. Then we compare the computation costs of the CSP side with two previous works, SWP [11] and
In PPTPS, it mainly derives from the Outsource phase and the Prove phase, where the CSP checks the tags’ validity and the CSP generates the integrity proof, respectively.

Let $l$ denote the challenged data blocks, according to the equations

$$\theta = g^r, \quad \omega = H(\theta), \quad x = r + \omega \sum_{j=1}^{l} d_j e_j,$$

$$\lambda = H(b_{l_0}|| \ldots || b_{l_l}|| t|| \theta|| x|| \beta),$$

we obtain that there are $l \cdot \text{Exp}_G$ operations, $(l - 1) \cdot \text{Mul}_G$ operations, $l \cdot \text{Mul}_{Z_p}$ operations, and $2 \cdot H_{Z_p}$ operations of the CSP side. It is shown in Table 3 that the computation cost in PPTPS is less than the computation cost in CPVPA and slightly more than the computation cost in SWP. Compared to SWP, the computation increment leads PPTPS to possess privacy-preserving and traceable properties. Therefore, this computation increment can be tolerated in practice.

Moreover, the computation cost of the auditor side derives from the Verify phase. We compute

$$\omega = H(\theta), \quad \lambda = H(b_{l_0}|| \ldots || b_{l_l}|| t|| \xi),$$

and checks whether

$$\beta = g^x \cdot \prod_{j=1}^{l} y_j^{e_j} \cdot \prod_{j=1}^{l} H(i|| fn|| y_j^{e_j}),$$

where $\beta = \prod_{j=1}^{l} s_j^{e_j}$. Therefore, there are $(3l + 1) \cdot \text{Exp}_G$ operations, $(3l - 1) \cdot \text{Mul}_G$ operations, $(l + 2) \cdot H_{Z_p}$ operations in this phase. The computation cost of the auditor side is described in Table 4. It is observed that the computation cost in PPTPS is no more than the computation costs in SWP and CPVPA. Fig 4 further demonstrates that PPTPS does not increase the computation cost. In the case of 200 challenged blocks, the auditor verifies the proof within 1 minute.

Finally, we vary the block size from 256 bits to 4096 bits. It is shown in Fig 5 that the computation costs in the Outsource, Verify, and Prove phases increase with the increment of the block size. Besides, the computation cost in the Outsource phase increases most dramatically. Note that the longer the block size is, the smaller the block number. Therefore, a tradeoff is made between the block size and the block number in pursuit of high efficiency.

Fig 2. The computation costs in the Outsource phase (ms).

https://doi.org/10.1371/journal.pone.0276212.g002
8.2 Communication cost

The communication cost involves two types. One type of communication cost derives from transferring the generated proof $\Gamma = (\theta, \alpha, \beta)$ between the CSP and the auditor, and the other type of communication cost derives from sending the generated hash value in each phase from
In PPTPS, the monetary costs are caused by conducting the corresponding transaction in Ethereum in each phase. For example, once a user generates and outsources data to the CSP, the user creates a transaction $T_1$. Next, an auditor generates an audit query and creates a transaction $T_2$ in the query phase. Moreover, the CSP generates a proof and creates a transaction $T_3$ in the proving phase. At last, the auditor verifies the proof and renders the result. Then the auditor creates a transaction $T_4$ to record the verification result.

Since each transaction on Ethereum needs computational effort to execute, each transaction requires a fee. In addition, Gas refers to the fee which is required to execute the specific operations on Ethereum. We compute the cost fee by $(\text{cost Gas}) \times (\text{gas price})$. The gas price is set as

### 8.3 monetary cost

In PPTPS, the monetary costs are caused by conducting the corresponding transaction in Ethereum in each phase. For example, once a user generates and outsources data to the CSP, the user creates a transaction $T_1$. Next, an auditor generates an audit query and creates a transaction $T_2$ in the query phase. Moreover, the CSP generates a proof and creates a transaction $T_3$ in the proving phase. At last, the auditor verifies the proof and renders the result. Then the auditor creates a transaction $T_4$ to record the verification result.

Since each transaction on Ethereum needs computational effort to execute, each transaction requires a fee. In addition, Gas refers to the fee which is required to execute the specific operations on Ethereum. We compute the cost fee by $(\text{cost Gas}) \times (\text{gas price})$. The gas price is set as
Fig 4. The computation cost of the auditor side.
https://doi.org/10.1371/journal.pone.0276212.g004

Fig 5. The computation cost for different block sizes.
https://doi.org/10.1371/journal.pone.0276212.g005
2 gwei, where 1 gwei = 10^{-9} ETH. Among all the transactions \( T_1 - T_4 \), the most gas costed by \( T_3 \) is 644169 gas. When writing this paper (September 21, 2022), 1 ETH is equivalent to $1430. Then we can observe that the most expensive transaction cost is about $1.84. Thus, the entire monetary cost for all the transactions is about $7.36, which is realistically affordable.

8.4 Discussion
PPTPS can be widely used in the field of public cloud storage. The timeliness enables PPTPS to keep track of critical activities for cloud storage, such as Outsource, Audit, Prove, and Verify. However, there are two necessary assumptions for PPTPS to work.

Assumption 1. The data owner does not collude with the CSP. moreover, the auditor is assumed to be a trusted third party and does not collude with the data owner and the CSP.

Assumption 2. PPTPS assumes that the transactions are all valid after posting to the blockchain. Miners in Ethereum are responsible for validating the transactions and packaging the valid transactions into Ethereum.

9 Conclusion and future work
In this paper, we construct a privacy-preserving publicly-auditable cloud storage system and present a traceable management method for the cloud storage system. Then we formally prove the security of PPTPS and analyze the resistance against different threats in detail. Finally, the experimental evaluation demonstrates its feasibility in computation, communication, and monetary cost. In the future, we will explore how to integrate blockchain with current cloud storage systems in depth, e.g., introducing the smart contract to provide more complicated and intelligent management for cloud storage systems. moreover, we will focus on the auditing problem based on searchable encryption in the follow-up work [41].
Author Contributions

Data curation: Mali Chen, Ting Luo, Li Chen.

Validation: Yangxin Wang, Yumeng Chen.

Writing – original draft: Li Li.

Writing – review & editing: Xiao Lan.

References

1. Shen WT, Qin J, Yu J, Hao R, Hu JK. Enabling Identity-Based Integrity Auditing and Data Sharing With Sensitive Information Hiding for Secure Cloud Storage. IEEE Transactions on Information Forensics and Security. 2019; 14(2):331–346. https://doi.org/10.1109/TIFS.2018.2850312

2. Liu MX, Fan K, Kumar N, He DB, Shi WB. Hash–balanced binary tree–based public auditing in vehicular edge computing and networks. International journal of communication systems. 2022; 35(12):27628–27640. https://doi.org/10.1002/dac.4134

3. Liu MX, Lu N, Yin JL, Cheng QF, Shi WB, Raymond CK. BA-Audit: Blockchain-Based Public Auditing for Aggregated Data Sharing in Edge-Assisted IoT. In: Security and Privacy in New Computing Environments. Springer International Publishing; 2022. p. 204–218.

4. Fan K, Bao ZJ, Liu MX, Vasilakos AK, Shi WB. Dredas: Decentralized, reliable and efficient remote outsourced data auditing scheme with blockchain smart contract for industrial IoT. Future Generation Computer Systems. 2020; 110:665–674. https://doi.org/10.1016/j.future.2019.10.014

5. Buldas A, Lipmaa H, Schoenmakers B. Optimally efficient accountable time-stamping. In: Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography. Springer-Verlag; 2000. p. 293–305.

6. Haber S, Stornetta WS. How to time-stamp a digital document. In: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology. Springer-Verlag; 1990. p. 437–455.

7. Cao S, Zhang G, Liu P, Zhang X, Neri F. Cloud-assisted secure eHealth systems for tamper-proofing EHR via blockchain. Information Sciences. 2019; 485:427–440. https://doi.org/10.1016/j.ins.2019.02.038

8. Zhang Y, Xu C, Lin X, Shen XS. Blockchain-based public integrity verification for cloud storage against procrastinating auditors. IEEE Transactions on Cloud Computing. 2019; 1–1. https://doi.org/10.1109/TCC.2018.2822718

9. Ateniese G, Fu K, Green M, Hohenberger S. Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Transactions on Information and System Security. 2006; 9:1–30. https://doi.org/10.1145/1127345.1127346

10. Juels A, Kaliski BS. Pors: Proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. Association for Computing Machinery; 2007. p. 584–597.

11. Shacham H, Waters B. Compact proofs of retrievability. Journal of cryptology. 2013; 26(3):442–483. https://doi.org/10.1007/s00145-012-9129-2

12. Wang C, Chow SS, Wang Q, Ren K, Lou W. Privacy-preserving public auditing for secure cloud storage. IEEE transactions on computers. 2013; 62(2):362–375. https://doi.org/10.1109/TC.2011.245

13. Chen F, Xiang T, Yang Y, Chow SS. Secure cloud storage meets with secure network coding. IEEE Transactions on Computers. 2016; 65(6):1936–1948. https://doi.org/10.1109/TC.2015.2456027

14. Zhang J, Dong Q. Efficient id-based public auditing for the outsourced data in cloud storage. Information Sciences. 2016; 1(14):343–344.

15. Li L, Liu J. Secacs: Enabling lightweight secure auditable cloud storage with data dynamics Journal of Information Security and Applications. 2020; 54:102545.

16. Zhang J, Yang Y, Chen Y, Chen F. A secure cloud storage system based on discrete logarithm problem. In: 2017 IEEE/ACM 25th International Symposium on Quality of Service. IEEE; 2017. p. 1–10.

17. Tian M, Ye S, Zhong H, Chen F, Gao C, Chen J. Publicly-verifiable proofs of storage based on the discrete logarithm problem. IEEE Access. 2019; 7:129071–129081. https://doi.org/10.1109/ACCESS.2019.2938976

18. Le A, Markopoulou A, Dimakis AG. Auditing for distributed storage systems. IEEE/ACM Transactions on Networking. 2016; 24:2182–2195. https://doi.org/10.1109/TNET.2015.2450761
19. Yang K, and Jia X. An efficient and secure dynamic auditing protocol for data storage in cloud computing. IEEE Transactions on Parallel and Distributed Systems. 2013; 24(9):1717–1726. https://doi.org/10.1109/TPDS.2012.278

20. Liu J, Huang K, Rong H, Wang H, Xian M. Privacy-preserving public auditing for regenerating-code-based cloud storage IEEE Transactions on Information Forensics and Security. 2015; 10(7):1513–1528.

21. Shen J, Shen J, Chen X, Huang X, Susilo W. An efficient public auditing protocol with novel dynamic structure for cloud data. IEEE Transactions on Information Forensics and Security. 2017; 12(10):2402–2415. https://doi.org/10.1109/TIFS.2017.2705620

22. Yu Y, Au MH, Ateniese G, Huang X, Susilo W, Dai Y, Min G. Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage. IEEE Transactions on Information Forensics and Security. 2017; 12(4):767–778. https://doi.org/10.1109/TIFS.2016.2615853

23. Liu C, Yang C, Zhang X, Chen J. External integrity verification for outsourced big data in cloud and iot: A big picture. Future Generation Computer Systems. 2015; 49:58–67. https://doi.org/10.1016/j.future.2014.08.007

24. Yu J, Ren K, Wang C, Varadharajan V. Enabling cloud storage auditing with key-exposure resistance. IEEE Transactions on Information Forensics and Security. 2015; 10(6):1167–1179. https://doi.org/10.1109/TIFS.2015.2400425

25. Ardagna CA, Asal R, Damiani E, and Vu QH. From security to assurance in the cloud: A survey. ACM Computing Surveys. 2015; 48:1–50. https://doi.org/10.1145/2767005

26. Demirel D, Krenn S, Lörünser T, Traverso G. Efficient and privacy preserving third party auditing for a distributed storage system. In: 2016 11th International Conference on Availability, Reliability and Security. IEEE; 2016. p. 88–97.

27. Wang B, Li B, Li H. Oruta: privacy-preserving public auditing for shared data in the cloud IEEE Transactions on Cloud Computing. 2014; 2(1):43–56.

28. Xue JT, Chunxiang XU, Zhao JN, Jianfeng MA. Identity-based public auditing for cloud storage systems against malicious auditors via blockchain. Science China(Information Sciences). 2019; 62(3):41–56.

29. Zhang Y, Xu C, Cheng N, Li H, Yang H, Shen X. Chronos+: An accurate blockchain-based time-stamping scheme for cloud storage. IEEE Transactions on Services Computing. 2020; 13(2):216–229.

30. Kim M, Yu S, Lee J, Park Y, Park Y. Design of secure protocol for cloud-assisted electronic health record system using blockchain. Sensors. 2020; 20(10):2913. https://doi.org/10.3390/s20102913 PMID: 32455635

31. Xie MD, Zhao QT, Hong HB, Chen C, Yu J. A novel blockchain-based and proxy-oriented public audit scheme for low performance terminal devices. Journal of Parallel and Distributed Computing. 2022; 169:58–71. https://doi.org/10.1016/j.jpdc.2022.06.002

32. Zhang C, Xu Y, Hu YP, Wu J, Ren J, Zhang YX. A Blockchain-Based Multi-Cloud Storage Data Auditing Scheme to Locate Faults. IEEE Transactions on Cloud Computing. 2021; 2168-7161:1–12. https://doi.org/10.1109/TCC.2021.3057771

33. Andreas M, Gavin W. Mastering Ethereum; 2018. p. 1–200.

34. Goyal R, Goyal V. Overcoming cryptographic impossibility results using blockchains. In: Theory of Cryptography. Springer International Publishing; 2017. p. 529–561.

35. Kiayias A, Panagiotakos G. Speed-security tradeoffs in blockchain protocols. IACR Cryptology ePrint Archive. 2015; 2015:1019.

36. Badertscher C, Maurer U, Tschudi D, Zikas V. Bitcoin as a transaction ledger: A composable treatment. In: Advances in Cryptology 2017. Springer International Publishing; 2017. p. 324–356.

37. Odlyzko A. Discrete Logarithms: The Past and the Future. Springer US; 2000. p. 59–75.

38. Li J, Yan H, Zhang Y. Certificateless public integrity checking of group shared data on cloud storage. IEEE Transactions on Services Computing. 2021; 14(1):71–81.

39. Pedersen TP. Non-interactive and information-theoretic secure verifiable secret sharing. In: Advances in Cryptology 91. Springer Berlin Heidelberg, 1992. p. 129–140.

40. Finlayson SG, LePendu P, Shah NH. Building the graph of medicine from millions of clinical narratives. Scientific data. 2014; 1(1):1–9. https://doi.org/10.1038/sdata.2014.32 PMID: 25977789

41. Gao X, Yu J, Chang Y, Wang HQ, Fan JX. Checking Only When It Is Necessary: Enabling Integrity Auditing Based on the Keyword with Sensitive Information Privacy for Encrypted Cloud Data. IEEE Transactions on Dependable and Secure Computing. 2021; 1941-0018:1–17. https://doi.org/10.1109/TDSC.2021.3106780