Accepted Manuscript

Outsourced dynamic provable data possession with batch update for secure cloud storage

Wei Guo, Hua Zhang, Sujuan Qin, Fei Gao, Zhengping Jin, Wenmin Li, Qiaoyan Wen

PII: S0167-739X(18)31986-1
DOI: https://doi.org/10.1016/j.future.2019.01.009
Reference: FUTURE 4702

To appear in: *Future Generation Computer Systems*

Received date : 3 September 2018
Revised date : 27 November 2018
Accepted date : 6 January 2019

Outsourced dynamic provable data possession with batch update for secure cloud storage

Wei Guo, Hua Zhang*, Sujuan Qin*, Fei Gao, Zhengping Jin*, Wenmin Li, Qiaoyan Wen

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

Abstract

With the advent of data outsourcing, how to efficiently verify the integrity of data stored at an untrusted cloud service provider (CSP) has become a significant problem in cloud storage. Provable data possession (PDP) is a model that allows clients or a trusted auditor to verify whether CSP possesses the outsourced data without downloading it. However, this model requires clients to tolerate non-negligible computation burden incurred by frequent verifications in private schemes, and does not provide any security assurances when client or/and auditor are dishonest. Therefore, it can not be directly transformed into a secure outsourced auditing scheme, where any one of three participants (i.e., CSP, client and auditor) may be dishonest and any two participants may be colluded with each other. In this paper, we propose an outsourced dynamic provable data possession (ODPDP) scheme, which migrates frequent auditing task to an external auditor to reduce clients' verification overhead, and simultaneously provides log audit mechanism with lower computation burden for clients to prevent from dishonest auditor. In addition, we propose a batch update algorithm that can perform and verify multiple update operations at once, avoiding repetitive calculations and transmissions. Security analysis proves that our scheme is secure in the enhanced threat model, and experimental results show that our scheme achieves high efficiency in terms of computation time and communication cost compared with existing outsourced auditing schemes.

Keywords: Cloud storage; provable data possession; outsourced auditing; dynamic update.

1. Introduction

Cloud computing has emerged as a predominant computing paradigm in recent years, attracting considerable attention of both industry and academia. It has the following five advantages: on-demand self-service, broad network access, resource pooling, rapid elasticity or expansion, and measured service [1]. As an important branch of cloud computing, cloud storage provides data outsourcing service for clients. However, data outsourcing means that data are no longer under clients' control, it introduces some security risks to the outsourced data, such as data breaches, data loss, etc [2]. So although cloud storage is promising, many potential clients are still unwilling to make the move unless the existing risks are eliminated. As a matter of fact, one of the major hurdles which make clients reluctant to use cloud storage is concern about the integrity of data stored at untrusted servers [3, 4].

To address the concern over data integrity and establish clients' confidence in cloud storage, it is critical to develop data integrity auditing by which clients are able to check the integrity of their data without downloading the entire data. To date, a number of solutions have been presented in the literature to guarantee data integrity [5–21]. All the state of the art schemes offer probabilistic guarantee of data integrity by sampling a random subset of all data blocks, which has been shown to be a practical strategy for verifying the integrity of large data sets [6]. In private auditing schemes [5, 8–10], clients need to frequently conduct verifications in order to ensure that their data stored at CSP are intact, which requires them to have network access and incurs non-negligible computation burden on them. To alleviate clients' burden incurred by frequent verification process, public auditing schemes were proposed [6, 7, 11–13, 17–21], which enable a trusted third party auditor (TPA) to perform integrity auditing task on behalf of clients.

However, existing public auditing schemes do not provide any security assurances if client or/and TPA are dishonest. An outsourced auditing scheme was proposed by Armknecht et al. [15] to address this problem, which can protect against any one dishonest participant and even against collusion of any two malicious participants. The scheme can also relieve clients from the frequent auditing process especially for the ones with resource-constrained device. Therefore, this novel business model is more readily adopted by cloud clients and can boost the development of cloud storage. Unfortunately, the scheme [15] is limited to static data, where the uploaded data can not be updated by client remotely. A dynamic outsourced auditing scheme was presented by Rao et al. [16] to solve this problem, which provides the same security guarantees as in [15] and can support dynamic update on the outsourced data by using Merkle tree. But it introduces some additional information for each tree node, such as status value and height value, leading to increased storage cost. In this case, the maintenance cost of Merkle tree will also be increased if data update occurs, as it needs to additionally update the above two values for each affected node.

*Corresponding author

Email addresses: zhanghua_288@bupt.edu.cn (Hua Zhang), qsujuan@bupt.edu.cn (Sujuan Qin), zhpjin@bupt.edu.cn (Zhengping Jin)

Moreover, the scheme [16] can just handle multiple update operations one by one, which wastes additional bandwidth and computation resources (cf. Section 3.2.5).

In fact, there already exists a dynamic PDP scheme based on skip list by Esiner et al. [10], which can handle multiple updates at once and achieve efficiency gains dramatically. Unfortunately, the scheme only considers private auditing, and can not be directly extended to support outsourced auditing. The reason is that the tag construction makes clients have to bear considerable computation effort of $O(kl)$[1] exponentiations when they check the auditor's any k log entries once—while this verification effort is almost equal to the effort of directly performing auditing k times with CSP. As a consequence, it can not satisfy the requirement in [15], which claims that log verification with auditor should be more computationally efficient than the direct verification of data with CSP.

In this paper, we propose an outsourced dynamic provable data possession (ODPDP) scheme to overcome the problems outlined above, which can perform and verify multiple update operations at once and support outsourced auditing simultaneously. The core thought behind our scheme is that the integrity of hash values for all data blocks is protected by the rank-based Merkle tree (RBMT), while the hash values together with the tags protect the integrity of data blocks. The main contributions of this work are summarized as follows:

(1) We propose a multi-leaf-authenticated (MLA) solution for RBMT, which is able to authenticate multiple leaf nodes and their indices all together without storing status value and height value. Based on MLA solution, we present a batch update algorithm that can perform and verify multiple update operations at once. The amortized price per authentication/update is decreased from $1+\log n$ to $1+\log(n/c)$, where n is the total number of data blocks and c is the number of challenged/updated data blocks.

(2) We propose an efficient homomorphic verifiable tag (E-HVT) based on BLS signature to meet with the requirement of [15], which can reduce clients' log verification effort in terms of exponentiations from $O(kl)$[1] to $O(1)$. We further propose a log audit mechanism by means of which client can check log files produced by auditor at a lower frequency to verify if auditor performed his auditing work honestly in the past.

(3) We describe a concrete ODPDP scheme that is secure in improved threat model (cf. Section 2.2), and can alleviate clients' verification overhead by migrating frequent auditing work to an external auditor. We implement the prototype of the ODPDP scheme, and experiments certify the high performance of our scheme.

The remainder of this paper is organized as follows. In Section 2, we introduce a notion of ODPDP scheme, followed by its threat model, then outline some building blocks exploited in our scheme. In Section 3, we propose a MLA solution for RBMT and then describe our construction in detail. Security analysis and performance analysis are given in Section 4 and 5, respectively. Related work is discussed in Section 6. Finally, we conclude the paper in Section 7.

[1] Where l is the number of challenged data blocks for each auditing between auditor and CSP, namely, each auditor's log entry involves l data blocks.

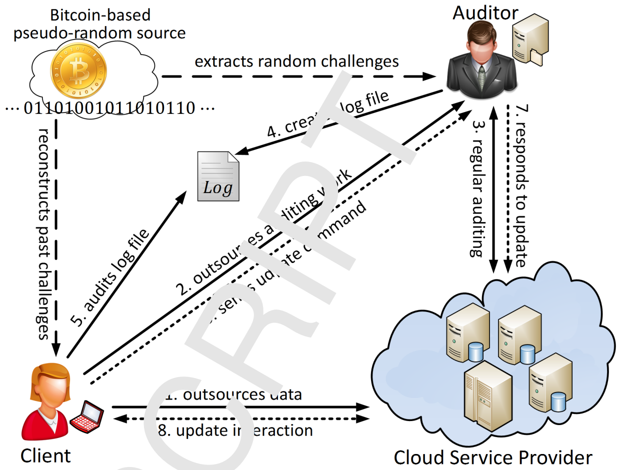


Fig. 1: Framework of ODPDP, where the solid arrows denote the auditing procedure which consists of steps 1–5, the dotted arrows denote the update process which consists of steps 6–8, the dashed arrows denote the challenges extraction and reconstruction. Followed by [15], the random challenges are extracted from an external Bitcoin-based pseudo-random source to prevent misbehavior and collusion.

2. System Model and Building Blocks

2.1. ODPDP Scheme

We start with a description of ODPDP framework, as depicted in Fig. 1. The following three participants are involved:

- Cloud Service Provider (CSP): an entity, who has configurable computation and storage resources, takes charge of data management and maintenance;
- Client: a data owner, who plans to outsource her data to CSP, is concerned about the data's integrity and may check whether the auditor did his work honestly;
- Auditor: an external auditor, who receives the auditing work from client, constantly monitors the integrity of client's data stored at CSP.

Then, we describe the workflow of Fig. 1 as follows: 1) client outsources her data to CSP; 2) client outsources an auditing work to the auditor; 3) by relying on functionality from Bitcoin [22], auditor regularly audits with CSP to check if the outsourced data is intact; 4) after each audit, auditor creates a log entry that records his auditing work at this point; 5) at any time, client can audit the log file to check whether auditor performed his auditing work honestly in the past; 6) client sends updated hash values to auditor; 7) auditor performs updates on the RBMT and sends an update proof to CSP for verification; 8) client also verifies the update proof received from CSP and sends updated data blocks to CSP.

Following [15, 16], a formal definition of ODPDP scheme is given below.

Definition 1. (ODPDP Scheme) *An ODPDP scheme is a collection of six protocols, which are described as below:*

- ***Setup***$(1^\kappa) \rightarrow$ {*client:* sk_c, vk_c, sk, pk; *auditor:* sk_a, vk_a; *CSP:* sk_{CSP}, vk_{CSP}} *is a randomized key generation protocol. It takes a security parameter* κ *as input, and outputs a matched signing-verifying key pair* (sk_P, vk_P) *for each participant P. In addition, it also returns a pair of secret and public keys* (sk, pk) *for client. For simplicity of expression, we suppose for each of subsequent protocols that an involved participant always takes as input the public keys of all participants and its own secret key.*

- ***Store***(*client:* M) $\rightarrow$ {*client:* $\mathbb{P}$, $\mathbb{C}$; *auditor:* $\mathbb{P}$, $\mathbb{C}$, T; *CSP:* $\mathbb{P}$, $\mathcal{M}$} *is an interactive protocol among three participants. It takes as input the keys of three participants and a data* M *held by client, then outputs the processed data* $\mathcal{M} = (M, \Sigma)$ *for CSP, where* Σ *is a tag vector of* M *generated by client with her secret key* sk*. It also outputs a RBMT* T *constructed over* M *for auditor. In addition, it outputs a public parameter* $\mathbb{P}$ *confirmed by three participants and a contract* $\mathbb{C}$ *signed between client and auditor.*

- ***AuditData***(*auditor:* Q, T; *CSP:* $\mathcal{M}$) $\rightarrow$ {*auditor:* dec_a, L} *is an interactive protocol run by auditor and CSP to convince the auditor that* M *is still intact at CSP. By leveraging functionality from Bitcoin, auditor extracts pseudorandom challenge* Q*, and sends it to CSP. Based on* Q *and* $\mathcal{M}$*, CSP computes a proof of data possession and responds auditor with it. Then auditor verifies the proof with* Q, T, pk*, and outputs a binary value* dec_a *which indicates whether or not he accepts the proof and a log entry* L *which records his auditing work.*

- ***AuditLog***(*client:* B; *auditor:* T, Λ) $\rightarrow$ {*client:* dec_c} *is an interactive protocol run by client and auditor to enable client to audit a log file* Λ *which consists of many log entries produced by auditor, the aim is to check if auditor was responsible to do his auditing work in the past. After receiving* B*, a random subset of indices of Bitcoin blocks released by client, auditor computes a proof of appointed logs based on* B, T, Λ *and sends it to client. Then client checks the proof, and outputs a binary value* dec_c *which indicates whether she accepts the proof or not. Note that this protocol should be much less frequent and more computationally efficient than* ***AuditData*** *protocol.*

- ***DynamicUpdate***(*client:* uc; *auditor:* T; *CSP:* $\mathcal{M}$) $\rightarrow$ {*True, False*} *is an interactive protocol among three participants to support provable update to the outsourced data. It takes as input client's update command* uc*, the tree* T *from auditor and the processed data* $\mathcal{M}$ *stored at CSP, and outputs True if the client's data is updated correctly, or False otherwise.*

- ***ImpartialArbitration***(*participant:* *evidences*) $\rightarrow$ {*True, False*} *is a protocol that is executed with the help of trusted arbitrator to deal with any disputes which may occur among three participants. The goal of this protocol is to protect the honest participant from malicious participants. It takes as input some evidences from one participant, and outputs True if the participant is honest, or False otherwise.*

2.2. Threat Model

Similar to existing work in the area [11, 15], we do not consider confidentiality of the data M in this paper, and more attention is paid to integrity of the data which is the core problem we study here. In what follows, we define the security goals of ODPDP scheme.

In outsourced auditing scheme, any one of the three involved participants may be dishonest, and even any two participants may be colluded with each other, which is different from traditional auditing schemes, where only CSP is dishonest. Therefore, more complex security problems need to be considered in outsourced auditing scheme. For example, the honest client should be protected from the collusion between CSP and auditor, or the honest auditor needs to defend against the malicious client and CSP etc. Followed by [15], to extend the existing threat model in [6, 7], the soundness of an ODPDP scheme is defined as follows.

Definition 2. (Soundness of ODPDP) *We say that an ODPDP scheme is sound if it satisfies the following three properties: authenticity, liability and extractability.*

Authenticity. To support provable update, ODPDP should guarantee the authenticity of the retrieved leaf nodes, both values stored in them and their indices.

Observe that if a scheme is secure against two malicious participants, then it automatically is secure against any one of them. Hence, it is sufficient to consider the following three cases where exactly one participant is honest.

Liability. For the first case, ODPDP should protect the honest auditor from the malicious client and CSP to minimize the auditor's liability in case of potential disputes, e.g., if the data is lost.

For the latter two cases, let us consider the following game between a challenger and an adversary $\mathcal{A}$ who corrupted CSP and auditor (or auditor and client):

- Setup: The challenger runs **Setup** protocol to generate all used keys, and provides $\mathcal{A}$ with secret keys of the corrupted participants and all public keys;

- Query: The adversary $\mathcal{A}$ is allowed to make query to a store oracle for any data M, and obtains the corresponding response from the challenger;

- Challenge: The challenger generates a challenge Q and requires the adversary $\mathcal{A}$ to provide a proof of possession for the data blocks specified by Q;

- Forge: The adversary $\mathcal{A}$ computes a proof of possession for the specified data blocks.

If the proof passes the challenger's verification, then the adversary $\mathcal{A}$ wins the game. In these protocols, the challenger plays the role of honest participant and the adversary $\mathcal{A}$ plays the role of the corrupted participants.

Extractability. If the adversary $\mathcal{A}$ wins the above game, then there exists an extractor that can recover the challenged data blocks in interaction with $\mathcal{A}$, which means that these data blocks are actually stored.

2.3. Building Blocks

We introduce some cryptographic building blocks used in our scheme. For notational simplicity, we use a symmetric bilinear map to describe our scheme, and our construction can be generally translated to an asymmetric setting [7] (cf. Section 3.2.7). Let $e : G \times G \to G_T$ be a bilinear map, where G and G_T are two multiplicative cyclic groups of prime order q [23]. Let g be a generator of G and κ be a security parameter. In addition, we define the following functions, as given by [6, 15].

- $H_2 : \{0,1\}^* \to \mathbb{Z}_q$;
- KeyGen: $\{0,1\}^\kappa \to \{sk, vk\}$;
- GetRandomness: $\Gamma \to \{0,1\}^{l_{hash}}$;
- PRBG: $\{0,1\}^\kappa \times \{0,1\}^{l_{hash}} \to \{0,1\}^*$;
- $\pi : \{0,1\}^\kappa \times \{0,1\}^{\log_2(n)} \to \{0,1\}^{\log_2(n)}$;
- $f : \{0,1\}^\kappa \times \{0,1\}^{\log_2(n)} \to \mathbb{Z}_q$.

More specifically, H_2 is a cryptographic hash function which maps an arbitrary length string uniformly to $\mathbb{Z}_q$. KeyGen is a key generation algorithm of a secure digital signature scheme, that takes the security parameter κ as input and outputs a pair of signing-verifying keys $\{sk, vk\}$. GetRandomness is a Bitcoin *getblockhash* function, which takes as input the current time $t \in \Gamma$ where Γ is a time set, and outputs the latest Bitcoin block's hash that is an uniformly random string in $\{0,1\}^{l_{hash}}$. PRBG is a pseudo-random bit generator which outputs long enough pseudo-random bits. In addition, a pseudo-random permutation π and a pseudo-random function f are also used to generate random indices i_η and coefficients a_η for each challenge respectively (cf. Section 3.2.3).

3. The Proposed Scheme

In this section, to achieve batch update and outsourced auditing, we propose a MLA solution that can authenticate multiple leaf nodes and their indices all together without storing status value and height value of tree nodes. Moreover, we describe a concrete instantiation of the proposed ODPDP scheme.

3.1. A Novel Multi-Leaf-Authenticated Solution

Inspired by [9], we develop a modification of Merkle tree [24] to support authentication of indices of leaf nodes, which we call a rank-based Merkle tree (RBMT). Concretely, the data field of each tree's node w in our scheme is composed of only two elements (r, h). To reduce storage and maintenance costs, we do not need to store the node's status value and height value which are necessary in [16]. The first element r stores the rank

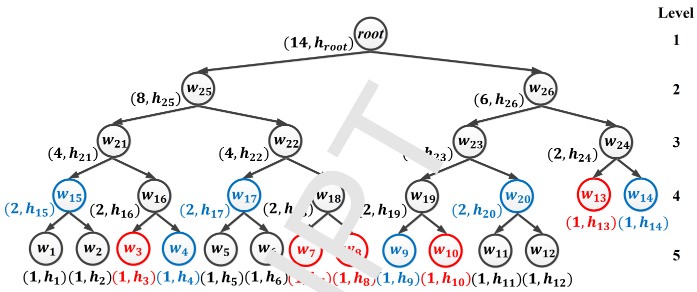


Fig. 2: The RBMT constructed over 14 data blocks, where the challenged nodes are marked with red, the necessary nodes needed for verification are marked with blue.

information which is the number of leaf nodes reachable from the node w. Particularly, $r = 1$ if w is a leaf node.

Next, let us explain the definition of the second element h. The outsourced data M held by client is comprised of n data blocks, namely $M = (m_1, m_2, \cdots, m_n)$. We bind the i-th data block m_i to the i-th leaf node w_i by storing the hash value of m_i at the node w_i. Therefore, all the leaf nodes of RBMT are already sorted by their indices in a left-to-right order. For each non-leaf node w, let $w.left$ and $w.right$ denote the left child and right child of that node, respectively, and $w.left.h$, $w.right.h$ denote the hash values of the two children. Let H_1 be a secure hash algorithm and $||$ denote concatenation, now the second element h is defined as follows

$$h = \begin{cases} H_2(m_i), \text{if } w \text{ is the } i\text{-th leaf node;} \\ H_1(r||w.left.h||w.right.h), \text{if } w \text{ is a non-leaf node.} \end{cases}$$

With the use of a collision-resistant hash function H_1, a RBMT can be constructed over the given n data blocks. Due to the dependency on all data blocks, knowing a Merkle root h_{root} (the hash value of the tree's root node) is sufficient for later integrity verification. In Fig. 2, we give an example of RBMT.

For the convenience of later description, we introduce some operations on a vector $\Theta = (\theta_1, \theta_2, \cdots, \theta_\tau)$, where each element $\theta_i \in \mathbb{Z}^+$ $(1 \le i \le \tau)$. For any $\xi \in \mathbb{Z}^+$, we define

$$\Theta \pm \xi = (\theta_1 \pm \xi, \theta_2 \pm \xi, \cdots, \theta_\tau \pm \xi);$$
$$\Theta(i) \pm \xi = (\theta_1, \cdots, \theta_{i-1}, \theta_i \pm \xi, \theta_{i+1}, \cdots, \theta_\tau);$$
$$\Theta(i\text{-after}) \pm \xi = (\theta_1, \cdots, \theta_i, \theta_{i+1} \pm \xi, \cdots, \theta_\tau \pm \xi);$$
$$\Theta(i, j) \pm \xi = (\cdots, \theta_{i-1}, \theta_i \pm \xi, \cdots, \theta_j \pm \xi, \theta_{j+1}, \cdots).$$

A strawman solution. When multiple leaf nodes are challenged, a straightforward solution is verifying these nodes one by one, just as the previous schemes [9, 11, 12, 14]. However, this solution leads the verifier to not only retrieve some repetitive nodes and unnecessary nodes but also perform some repetitive hash calculations, which incurs additional communication and computation costs. In the example of Fig. 2, the proofs of w_3 and w_7 are $\Omega_3 = \{w_{26}, w_{22}, w_{15}, w_4, w_3\}$ and $\Omega_7 = \{w_{26}, w_{21}, w_{17}, w_8, w_7\}$, separately. If the verifier authenticates w_3 and w_7 individually, then the same node w_{26} will be retrieved twice and the hash values of w_{25}, *root* will be calculated

Algorithm 1 GenMultiProof(T, C)

Input: the RBMT T and the challenged index vector $C = (i_1, i_2, \cdots, i_c)$, where $i_\alpha < i_\beta$ for $1 \le \alpha < \beta \le c$.
Output: the corresponding multi-proof $\sqcup_p$ for all the c challenged leaf nodes.

```
initialize empty stacks ⊔_s, ⊔_p;
push (root, C) to ⊔_s;
for j = 1 to c do
  (cn, C) = pop(⊔_s); {where cn denotes current node}
  while cn.r ≠ 1 do
    initialize empty vectors C_l, C_r;
    for each element C[γ] in C do
      if C[γ] ≤ cn.left.r then
        add C[γ] to C_l;
      else {C[γ] > cn.left.r}
        add C[γ] to C_r;
      end if
    end for
    if C_l ≠ NULL and C_r ≠ NULL then
      push (cn.right, C_r − cn.left.r) to ⊔_s;
      push (I, NULL) to ⊔_p;
      cn = cn.left; C = C_l;
    else if C_l ≠ NULL and C_r == NULL then
      push (L, cn.right) to ⊔_p;
      cn = cn.left; C = C_l;
    else {C_l == NULL and C_r ≠ NULL}
      push (R, cn.left) to ⊔_p;
      C_r = C_r − cn.left.r;
      cn = cn.right; C = C_r;
    end if
  end while
  if cn.r == 1 then
    push (E, cn) to ⊔_p;
  end if
end for
return ⊔_p;
```

w_{13}	E	w_{13}
w_{24}	L	w_{14}
w_{10}	E	w_{10}
w_{19}	R	w_9
w_{23}	L	w_{20}
w_{26}	I	NULL
w_8	E	w_8
w_7	E	w_7
w_{18}	I	NULL
w_{22}	R	w_{17}
w_3	E	w_3
w_{16}	L	w_4
w_{21}	R	w_{15}
w_{25}	I	NULL
$root$	I	NULL

(a) The multi-proof $\sqcup_p$.

w'_{13}	E	w'_{13}
w^*_{24}	L	w_{14}
w_9	E	w_9
w^*_{23}	L	w_{20}
w^*_{26}	I	NULL
w'_8	E	w'_8
w^*_{22}	R	w_{17}
w^*_3	E	w^*_3
w^*_{16}	L	w_4
w^*_{21}	R	w_{15}
w^*_{25}	I	NULL
$root^*$	I	NULL

(b) The updated multi-proof $\sqcup^*_p$.

Fig. 3: The current node *cn* in each iteration is listed on the left of the stacks to improve readability. The flag E denotes *cn* is a challenged node, the flag I denotes *cn* is an intersection node, the flag L/R means that the challenged nodes indexed by the current vector C can be found by following the left/right pointer from *cn*.

twice. In fact, the nodes w_{21}, w_{22} are unnecessary for verifying w_3 and w_7 simultaneously. This situation will be exacerbated when the number of challenged leaf nodes is increasing. Therefore, the solution is not a cost-efficient way in ODPDP scheme if client wants to audit many log entries (involving too many leaf nodes) in order to check auditor's auditing work.

To remedy this problem, we propose a MLA solution that can verify multiple leaf nodes and their indices all together. Consequently, only the necessary nodes needed for verification should be retrieved, and only the necessary hash calculations needed for verification should be performed. More specifically, the prover generates a multi-proof $\sqcup_p$ by running Algorithm 1 GenMultiProof(T, C), where C is a challenged index vector launched by the verifier. This algorithm starts from the pair ($root, C$), where $root$ is the root node of T. Then, in each iteration of the outer for-loop, this algorithm pushes a number of flag-node pairs to the stack $\sqcup_p$ until the left-most challenged node is included in it. There are four possible flags (such as E, I, L and R) in this algorithm, which meanings are explained in the caption of Fig. 3. For each iteration of the outer for-loop except the first one, it starts from the top element in the stack $\sqcup_s$, which stores the right child of intersection node and the updated C. Here, the intersection node means that the current challenged nodes can be found following both the left pointer and right pointer of it. For example, given $C = (3, 7, 8, 10, 13)$ in Fig. 2 (page 4), the corresponding multi-proof $\sqcup_p$ is shown in Fig. 3(a), where every necessary node appears just once.

After receiving the multi-proof $\sqcup_p$ from the prover, the verifier executes Algorithm 2 VerMultiProof($\sqcup_p, n, h_{root}, C$) to verify the leaf nodes specified by her challenge C all together. This algorithm goes in a top-down order on the stack $\sqcup_p$, which is contrary to that of multi-proof generation in Algorithm 1. This order corresponds to a right-to-left and bottom-up traverse in the tree T, so that hash calculations follow all the dependency relationships. For the received multi-proof $\sqcup_p$, Algorithm 2 iteratively computes three values V, r, h in every for-loop. If $\sqcup_p$ is a correct multi-proof for the leaf nodes indexed by C, then the following three properties hold for the values V, r, h computed in the last iteration of for-loop:

- Value V is equal to the challenged index vector C;
- Value r is equal to the total number of data blocks n;
- Value h is equal to the Merkle root h_{root} of T.

Note that the two algorithms also go well for a single leaf node.

Now we analyze the performance of our MLA solution. First, the MLA solution exhibits the worst-case performance only if the challenged leaf nodes are uniformly distributed among all the leaf nodes, since the number of repetitive and unnecessary nodes avoided by the MLA solution is smallest under this case. Second, for two uniformly distributed leaf nodes, the amortized price per authentication is equal to the price of verifying the first node in the left subtree, namely, $1 + \log(n/2)$. Analogously, the amortized price is $1 + \log(n/4)$ for four uniformly distributed leaf nodes, and so on. Finally, we argue that if the number of

Algorithm 2 VerMultiProof($\sqcup_p, n, h_{root}, C$)

Input: the multi-proof $\sqcup_p$, the total number of data blocks n, the Merkle root h_{root} and the challenged index vector $C = (i_1, i_2, \cdots, i_c)$.
Output: Accept or Reject.

```
start = c; end = c; δ = c;
initialize empty stacks ⊔_i, ⊔_r and ⊔_h;
initialize verification index vector V = (1, 1, ··· , 1), which
  has c elements;
  {let ⊔_p.top point to the top element in ⊔_p}
for k = ⊔_p.top to 1 do
  (d_k, v_k) = pop(⊔_p);
  if d_k == E then
    if δ ≠ c then
      push (start, end), r and h to ⊔_i, ⊔_r and ⊔_h, respec-
        tively;
      start = start − 1; end = start;
    end if
    δ = δ − 1;
    r = v_k.r; h = v_k.h;
  else if d_k == I then
    (tempStart, end) = pop(⊔_i);
    V = V(tempStart, end) + r;
    r = r + pop(⊔_r);
    h = H_1(r||h||pop(⊔_h));
  else if d_k == R then
    V = V(start, end) + v_k.r;
    r = v_k.r + r;
    h = H_1(r||v_k.h||h);
  else {d_k == L}
    r = r + v_k.r;
    h = H_1(r||h||v_k.h);
  end if
end for
if V == C and r == n and h == h_root then
  return Accept;
else
  return Reject;
end if
```

challenged leaf nodes is c ($1 \leq c \leq n$), then the amortized price per authentication of MLA solution is $1 + \log(n/c)$ in the worst case. Note that our solution achieves a constant complexity $O(1)$ in the case of $c = n$. However, the performance of the strawman solution is always $1 + \log n$ no matter how many leaf nodes are challenged.

3.2. Outsourced Dynamic Provable Data Possession Scheme

3.2.1. Setup Protocol

Each participant $P \in$ {CSP, client, auditor} performs KeyGen to obtain a secret signing key sk_P and a public verifying key vk_P. In addition, client samples $s + 1$ random elements $\alpha_1, \alpha_2, \cdots, \alpha_s, x \in \mathbb{Z}_q$ and computes $g_1 = g^{\alpha_1}, g_2 = g^{\alpha_2}, \cdots, g_s = g^{\alpha_s}, y = g^x \in G$. After that, client samples a random element $\lambda \in G$, now the client's secret key and public key are denoted as $sk = (\alpha_1, \alpha_2, \cdots, \alpha_s, x)$ and $pk = (g, \lambda, g_1, g_2, \cdots, g_s, y)$, respectively.

Table 1: Workflow of the **Store** Protocol

Client		**CSP**
1. Disperse M into $\{m_{iz}\}_{1\leq i\leq n}^{1\leq z\leq s}$; 2. Compute $h_i = H_2(m_i)$ for each $1 \leq i \leq n$, construct T; 3. Compute $\sigma_i = (\lambda^{h_i} \cdot g^{\sum_{z=1}^{s} \alpha_z m_{iz}})^x$ for each $1 \leq i \leq n$;		
	$\mathcal{M}=\{M, \Sigma\}$, $\mathrm{Sig}_{sk_c}(\mathcal{M})$ $\longrightarrow$	
		4. Verify $\mathrm{Sig}_{sk_c}(\mathcal{M})$, if succeed, store $\mathcal{M}$;
		Auditor
	T, $\mathrm{Sig}_{sk_c}(T)$ $\longrightarrow$	
		5. Verify $\mathrm{Sig}_{sk_c}(T)$, if succeed, store T;
	6. Three participants sign $\mathbb{P} = \{n, h_{root}\}$; 7. Client and Auditor sign $\mathbb{C} = \{\mathrm{BI}, \mathrm{F}, l\}$.	

3.2.2. Store Protocol

The outsourced data held by client is $M = (m_1, m_2, \cdots, m_n)$, where each data block consists of s sectors. More precisely, each data block has the form $m_i = m_{i1}||m_{i2}|| \cdots ||m_{is}$ ($1 \leq i \leq n$) such that each sector $m_{iz} \in \mathbb{Z}_q$ ($1 \leq z \leq s$), where $||$ denotes concatenation. The workflow of the **Store** protocol is shown in Table 1.

Constructing RBMT: With all data blocks, client first computes hash values $h_i = H_2(m_i)$ ($1 \leq i \leq n$). Then she constructs RBMT T on top of the ordered hash values, meaning that each leaf node w_i stores the corresponding hash value h_i.

Computing EHVT: Based on g, λ and secret key sk, client computes

$$\sigma_i = (\lambda^{h_i} \cdot g^{\sum_{z=1}^{s} \alpha_z m_{iz}})^x \in G \ (1 \leq i \leq n),$$

where the effort of exponentiations is independent of s, the number of sectors per block. Thus, this tag construction can improve **Store** performance considerably compared to the following traditional method [7, 12]

$$\sigma_i = (h_i \cdot \prod_{z=1}^{s} g_z^{m_{iz}})^x \in G \ (1 \leq i \leq n), \quad (1)$$

where the effort of exponentiations is positively correlated with the parameter s. Then client generates the processed data $\mathcal{M} = \{M, \Sigma\}$, where $\Sigma = (\sigma_1, \sigma_2, \cdots, \sigma_n)$.

Outsourcing data: Client sends $\mathcal{M}$ and its signature $\mathrm{Sig}_{sk_c}(\mathcal{M})$ to CSP. The latter verifies the signature $\mathrm{Sig}_{sk_c}(\mathcal{M})$, if the verification is not passed, then CSP rejects $\mathcal{M}$, which means that the client is malicious; otherwise, CSP accepts $\mathcal{M}$ by responding client with its reception and signature.

Outsourcing auditing work: Once the verification on CSP's signature is passed, client outsources auditing work to auditor by sending T with her signature $\mathrm{Sig}_{sk_c}(T)$. After that, the signature $\mathrm{Sig}_{sk_c}(T)$ will be verified by auditor, if the verification is passed, then auditor accepts T and responds client with his reception and signature; otherwise, auditor rejects T, which also means that the client is malicious.

Agreeing parameters: All participants need to agree on a public parameter $\mathbb{P} = \{n, h_{root}\}$, where n is the total number of data block and h_{root} is the Merkle root of T. In other words, three participants must sign $\mathbb{P}$ to reach an agreement. In addition, client and auditor need to further agree on a contract $\mathbb{C} = \{\text{BI}, \text{F}, l\}$ that specifies the checking policy for auditor. Specifically, BI denotes a Bitcoin block index from which the auditing work will start, F indicates the frequency at which the auditor launches a challenge, l dictates the number of challenged data blocks for each checking. Similarly, client and auditor must sign $\mathbb{C}$ to confirm the contract.

Now client deletes $\mathcal{M}$ and T from its local storage, she only maintains a constant amount of metadata. Note that the auditor can use public key pk to check the integrity of client's data, so there is not need for auditor to generate the parallel tags as in [15]. Consequently, the high communication cost of downloading the entire data from CSP to auditor is avoided in our **Store** protocol, and the expensive zero knowledge proof (ZKP) interaction between client and auditor is also avoided. In addition, client does not need to verify the auditor's tag, which is required in [15] and incurs considerable computation overhead on client.

3.2.3. AuditData Protocol

Our scheme leverages the Bitcoin blockchain as a time-dependent pseudo-random source to generate periodic challenges, its security and randomness have been demonstrated in [15]. The workflow of the **AuditData** protocol is shown in Table 2. Concretely, by inputting the time $t \in \Gamma$, auditor first runs GetRandomness to obtain a hash value $hash^{(b)} \in \{0,1\}^*$ of the latest block (let b denote the index of this block) that has appeared since time t in Bitcoin blockchain [25]. Then PRBG is invoked on the input $hash^{(b)}$ to obtain long enough pseudorandom bits, that will be sequentially used by auditor to select a pair of keys $k_\pi^{(b)}, k_f^{(b)}$. At last, auditor generates a challenge $Q^{(b)} = \{b, k_\pi^{(b)}, k_f^{(b)}\}$ and sends it to CSP, where the block b corresponds to the time t. Due to the property of Bitcoin blockchain, the challenge $Q^{(b)}$ is unpredictable to CSP and undeniable to auditor, respectively.

Upon receiving the challenge $Q^{(b)}$, CSP first computes the challenged indices and coefficients as follows

$$i_\eta = \pi_{k_\pi^{(b)}}(\eta),\ a_\eta = f_{k_f^{(b)}}(\eta)\ (1 \le \eta \le l). \tag{2}$$

Then to prove the integrity of the challenged data blocks, CSP computes the proof of data possession as follows

$$\mu_z^{(b)} = \sum_{\eta=1}^{l} a_\eta m_{i_\eta z} \in \mathbb{Z}_q\ (1 \le z \le s),\ \sigma^{(b)} = \prod_{\eta=1}^{l} \sigma_{i_\eta}^{a_\eta} \in G.$$

Finally, CSP responses the auditor with the proof $\rho^{(b)} = \{\mu_1^{(b)}, \mu_2^{(b)}, \cdots, \mu_s^{(b)}, \sigma^{(b)}\}$ and its signature $\text{Sig}_{sk_{\text{CSP}}}(\rho^{(b)})$.

Table 2: Workflow of the **AuditData** Protocol

Auditor		**CSP**
1. Generate a challenge $Q^{(b)} = \{b, k_\pi^{(b)}, k_f^{(b)}\}$;		
	$\xrightarrow{Q^{(b)}=\{b,k_\pi^{(b)},k_f^{(b)}\}}$	
		2. Compute $\{i_\eta, a_\eta\}_{1\le\eta\le l}$; 3. Compute $\mu_z^{(b)} = \sum_{\eta=1}^{l} a_\eta m_{i_\eta z}$ for each $1 \le z \le s$; 4. Compute $\sigma^{(b)} = \prod_{\eta=1}^{l} \sigma_{i_\eta}^{a_\eta}$;
	$\xleftarrow[\text{Sig}_{sk_{\text{CSP}}}(\rho^{(b)})]{\rho^{(b)}=\{\mu_1^{(b)},\cdots,\mu_s^{(b)},\sigma^{(b)}\}}$	
5. Verify $\text{Sig}_{sk_{\text{CSP}}}(\rho^{(b)})$, if succeed, continue; 6. Compute $\{i_\eta, a_\eta\}_{1\le\eta\le l}$; 7. Compute $h^{(b)} = \lambda^{\sum_{\eta=1}^{l} a_\eta h_{i_\eta}}$; 8. Verify $e(\sigma^{(b)}, g) \overset{?}{=} e(h^{(b)} \cdot \prod_{z=1}^{s} g_z^{\mu_z^{(b)}}, y)$; 9. Create $L^{(b)} = \{t, Q^{(b)}, h^{(b)}, \rho^{(b)}, \text{Sig}_{sk_{\text{CSP}}}(\rho^{(b)})\}$.		

If the verification on $\text{Sig}_{sk_{\text{CSP}}}(\rho^{(b)})$ is passed, then auditor verifies the correctness of $\rho^{(b)}$. First, auditor computes the challenged indices and coefficients by using $Q^{(b)}$ as in Eq. (2). Second, with the corresponding hash values stored in his local T, auditor computes the value as below

$$h^{(b)} = \lambda^{\sum_{\eta=1}^{l} a_\eta h_{i_\eta}} \in G.$$

Third, auditor verifies the proof $\rho^{(b)}$ by checking the following equality

$$e(\sigma^{(b)}, g) \overset{?}{=} e(h^{(b)} \cdot \prod_{z=1}^{s} g_z^{\mu_z^{(b)}}, y). \tag{3}$$

If the equality does not hold, meaning that at least one of the challenged data blocks has been lost or corrupted, then auditor informs client of this abnormal situation immediately. Otherwise, auditor is assured that the challenged data blocks are intact. Lastly, auditor creates the following log entry that records his auditing work

$$L^{(b)} = \{t, Q^{(b)}, h^{(b)}, \rho^{(b)}, \text{Sig}_{sk_{\text{CSP}}}(\rho^{(b)})\},$$

and saves it in his local log file Λ.

Based on the properties of bilinear map, the correctness of Eq. (3) can be elaborated as follows

$$\begin{aligned} e(\sigma^{(b)}, g) &= e(\prod_{\eta=1}^{l} (\lambda^{h_{i_\eta}} \cdot g^{\sum_{z=1}^{s} \alpha_z m_{i_\eta z}})^{a_\eta}, g^x) \\ &= e(\prod_{\eta=1}^{l} (\lambda^{a_\eta h_{i_\eta}}) \cdot \prod_{\eta=1}^{l} (\prod_{z=1}^{s} g_z^{a_\eta m_{i_\eta z}}), y) \\ &= e(\lambda^{\sum_{\eta=1}^{l} a_\eta h_{i_\eta}} \cdot \prod_{z=1}^{s} g_z^{\sum_{\eta=1}^{l} a_\eta m_{i_\eta z}}, y) \\ &= e(h^{(b)} \cdot \prod_{z=1}^{s} g_z^{\mu_z^{(b)}}, y). \end{aligned}$$

3.2.4. AuditLog Protocol

Outsourced auditing scheme must resist against malicious auditor, which is not captured in traditional auditing schemes. To this end, the **AuditLog** protocol (i.e., log audit mechanism) is designed to enable client to check if the auditor did his work honestly at any point in time. However, it should be noted that this auditing should be less frequent and more computationally efficient when compared to the direct auditing of data. The workflow of the **AuditLog** protocol is shown in Table 3.

As pointed in [15], client can audit the most recent log entry generated by auditor to minimize her checking work, because this does mirror the latest state of integrity for the monitored data. More generally, client can audit any subset of the log file in a batch way, no matter how many log entries are produced. More specifically, client chooses a random subset B of indices of Bitcoin blocks, and sends it to auditor.

Once receiving B, auditor finds $Q^{(b)}$, $h^{(b)}$ and $\rho^{(b)}$ from his log file Λ for each $b \in B$, and computes

$$h^{(B)} = \prod_{b\in B} h^{(b)} \in G,\ \sigma^{(B)} = \prod_{b\in B} \sigma^{(b)} \in G,$$

$$\mu_z^{(B)} = \sum_{b\in B} \mu_z^{(b)} \in \mathbb{Z}_q \ (1 \le z \le s).$$

In addition, for each $b \in B$, auditor reads $k_\pi^{(b)}$ from $Q^{(b)}$, and computes the challenged indices i_η $(1 \le \eta \le l)$ by invoking $\pi_{k_\pi^{(b)}}(\eta)$. After eliminating the repetitive indices, the last ordered challenge index vector is denoted by $C = (i_1, i_2, \cdots, i_c)$. Then auditor runs Algorithm 1 GenMultiProof(T, C) to obtain the corresponding multi-proof $\sqcup_p$. At last, auditor generates the proof of appointed logs as below

$$\rho^{(B)} = \{\sqcup_p, h^{(B)}, \mu_1^{(B)}, \mu_2^{(B)}, \cdots, \mu_s^{(B)}, \sigma^{(B)}\},$$

and sends it to client with his signature $\mathrm{Sig}_{sk_a}(\rho^{(B)})$.

After verifying the signature $\mathrm{Sig}_{sk_a}(\rho^{(B)})$, for each $b \in B$, client first invokes PRBG$(hash^{(b)})$ to get $Q^{(b)}$, and reconstructs the challenged indices and coefficients i_η, a_η $(1 \le \eta \le l)$ as in Eq. (2). Then client verifies the correctness of $\sqcup_p$ by calling Algorithm 2 VerMultiProof$(\sqcup_p, n, h_{root}, C)$, where C can be obtained by utilizing her own constructed indices for all $b \in B$. If the verification is passed, which means that all the challenged leaf nodes w_{i_j} $(1 \le j \le c)$ in $\sqcup_p$ including their indices are authenticated, then the corresponding hash value h_{i_j} stored in leaf node w_{i_j} can be accepted by client. Otherwise, client rejects $\sqcup_p$, meaning that the auditor is malicious. Finally, with λ and all authenticated h_{i_j}, client verifies $h^{(B)}$ by checking the following equation

$$h^{(B)} \stackrel{?}{=} \lambda^{\sum_{b\in B}(\sum_{\eta=1}^{l} a_\eta h_{i_\eta})}. \quad (4)$$

Here, the effort of exponentiations is independent of the two parameters $k = |B|$ and l due to our EHVT. Thus, this reduces the effort from $O(kl)$ exponentiations to $O(1)$ exponentiation when compared to the equation

$$h^{(B)} \stackrel{?}{=} \prod_{b\in B}(\prod_{\eta=1}^{l} h_{i_\eta}^{a_\eta}),$$

Table 3: Workflow of the **AuditLog** Protocol

Client		Auditor
1. Choose a random subset B of indices of Bitcoin blocks;	the random subset B $\longrightarrow$	
		2. Compute $h^{(B)} = \prod_{b\in B} h^{(b)}$; 3. Compute $\sigma^{(B)} = \prod_{b\in B} \sigma^{(b)}$; 4. Compute $\mu_z^{(B)} = \sum_{b\in B} \mu_z^{(b)}$ for each $1 \le z \le s$; 5. Generate $\sqcup_p$ by running Algorithm 1;
	$\{\sqcup_p, h^{(B)}, \mu_1^{(B)}, \cdots, \mu_s^{(B)}, \sigma^{(B)}\}$, $\mathrm{Sig}_{sk_a}(\rho^{(B)})$ $\longleftarrow$	
6. Verify $\mathrm{Sig}_{sk_a}(\rho^{(B)})$, if succeed, continue; 7. Verify $\sqcup_p$ by running Algorithm 2; 8. If succeed, verify $h^{(B)} \stackrel{?}{=} \lambda^{\sum_{b\in B}(\sum_{\eta=1}^{l} a_\eta h_{i_\eta})}$; 9. If succeed, verify $\sigma^{(B)} \stackrel{?}{=} (h^{(B)} \cdot g^{\sum_{z=1}^{s} \alpha_z \mu_z^{(B)}})^x$.		

which corresponds to the traditional tag construction in Eq. (1). If this verification fails, client rejects $h^{(B)}$, which also means that auditor is malicious. Otherwise, client checks the last equation by using her secret key sk and the verified $h^{(B)}$

$$\sigma^{(B)} \stackrel{?}{=} (h^{(B)} \cdot g^{\sum_{z=1}^{s} \alpha_z \mu_z^{(B)}})^x. \quad (5)$$

If the above Eq. (5) does not hold, client rejects $\mu_1^{(B)}, \mu_2^{(B)}, \cdots, \mu_s^{(B)}, \sigma^{(B)}$, manifesting that auditor has colluded with CSP and data has been corrupted in CSP. Otherwise, client can rest assured that auditor was honest to audit CSP for all the past challenged data blocks appointed by B. The correctness of Eq. (5) can be elaborated as below

$$\begin{aligned}\sigma^{(B)} &= \prod_{b\in B}\prod_{\eta=1}^{l} \sigma_{i_\eta}^{a_\eta} \\ &= \prod_{b\in B}\prod_{\eta=1}^{l} (\lambda^{h_{i_\eta}} \cdot g^{\sum_{z=1}^{s} \alpha_z m_{i_\eta z}})^{a_\eta x} \\ &= (\prod_{b\in B} \lambda^{\sum_{\eta=1}^{l} a_\eta h_{i_\eta}} \cdot g^{\sum_{z=1}^{s} \alpha_z (\sum_{\eta=1}^{l} a_\eta m_{i_\eta z})})^x \\ &= (\lambda^{\sum_{b\in B}(\sum_{\eta=1}^{l} a_\eta h_{i_\eta})} \cdot g^{\sum_{z=1}^{s} \alpha_z (\sum_{b\in B} \mu_z^{(b)})})^x \\ &= (h^{(B)} \cdot g^{\sum_{z=1}^{s} \alpha_z \mu_z^{(B)}})^x.\end{aligned}$$

Note that, to reduce computation cost as much as possible, we propose the client can accomplish the last verification with her secret key sk as in Eq. (5), which is more appropriate for the outsourced auditing scheme. More precisely, it not only avoids two pair operations but also reduces s multiplications and s exponentiations on group G to 1 and 2 respectively, when compared to the following equation

$$e(\sigma^{(B)}, g) \stackrel{?}{=} e(h^{(B)} \cdot \prod_{z=1}^{s} g_z^{\mu_z^{(B)}}, y).$$

Algorithm 3 BatchUpdate(C, U, M^*)

Input: the challenged index vector $C = (i_1, i_2, \cdots, i_c)$, the update operation vector $U = (U[1], U[2], \cdots, U[c])$ and the new data block vector $M^* = (m^*_{i_1}, m^*_{i_2}, \cdots, m^*_{i_c})$.
{note that if $U[j]$ == delete then $m^*_{i_j}$ = NULL}

Output: True or False.

Client:

1: **for** $j = c$ **to** 1 **do**
2: **if** $U[j]$ == modify **or** insert **then**
3: $h^*_{i_j} = H_2(m^*_{i_j})$; $\sigma^*_{i_j} = (\lambda^{h^*_{i_j}} \cdot g^{\sum_{z=1}^{s} \alpha_z m^*_{i_j z}})^x$;
{where $m^*_{i_j} = m^*_{i_j 1} \| m^*_{i_j 2} \| \cdots \| m^*_{i_j s}$}
4: **else** {$U[j]$ == delete}
5: $h^*_{i_j}$ = NULL; $\sigma^*_{i_j}$ = NULL;
6: **end if**
7: **end for**
8: send update command $uc = (C, U, H^*)$ and her signature $\mathrm{Sig}_{sk_c}(uc)$ to auditor and CSP;
{where $H^* = (h^*_{i_1}, h^*_{i_2}, \cdots, h^*_{i_c})$}

Auditor:

9: obtain multi-proof $\sqcup_p$ by calling Algorithm 1;
10: update all challenged leaf nodes by calling Algorithm 4, and output T', W;
11: obtain n^*, C^* and $\sqcup^*_p$ by calling Algorithm 5;
12: update other affected nodes in a way similar to Algorithm 2 by inputting T', $\sqcup^*_p$ and C^*, and output the final tree T^* and its Merkle root h_{root^*};
13: generate update proof $up = (\sqcup_p, h_{root^*})$, and send up and his signature $\mathrm{Sig}_{sk_a}(up)$ to CSP;

CSP:

14: verify update proof up by calling Algorithm 6; if this verification passes, then send up and its signature $\mathrm{Sig}_{sk_{CSP}}(up)$ to client; otherwise, output False;

Client:

15: verify up by calling Algorithm 6; if the verification is passed, then generate update information $ui = (M^*, \Sigma^*)$ and send it to CSP with her signature $\mathrm{Sig}_{sk_c}(ui)$; otherwise, output False; {where $\Sigma^* = (\sigma^*_{i_1}, \sigma^*_{i_2}, \cdots, \sigma^*_{i_c})$}

CSP:

16: **for** $j = c$ **to** 1 **do**
17: **if** $U[j]$ == modify **and** $h^*_{i_j} == H_2(m^*_{i_j})$ **then**
18: replace m_{i_j}, σ_{i_j} with $m^*_{i_j}, \sigma^*_{i_j}$, respectively;
19: **else if** $U[j]$ == insert **and** $h^*_{i_j} == H_2(m^*_{i_j})$ **then**
20: insert $m^*_{i_j}, \sigma^*_{i_j}$ before m_{i_j}, σ_{i_j}, respectively;
21: **else** {$U[j]$ == delete}
22: delete m_{i_j} and σ_{i_j}, respectively;
23: **end if**
24: **end for**

CSP, Client and Auditor:

25: sign the updated $\mathbb{P}^* = \{n^*, h_{root^*}\}$ jointly;
26: **return** True;

3.2.5. DynamicUpdate Protocol

The three kinds of update operations in our scheme are defined as follows: modification of the i-th data block, deletion of the i-th data block, and insertion of a new data block before the i-th data block. Owing to the fact that insertion of a new data block after the i-th data block is much similar to the third operation, thus is omitted here. To prevent malicious behavior and collusion, we propose the involved three participants need to reach a consistency of update operations.

Algorithm 4 UpdateLeafNode(T, uc)

Input: the tree T and the update command uc.
{where $uc = (C, U, H^*)$}

Output: the updated tree T' and the new node vector W.

for $j = c$ **to** 1 **do**
if $U[j]$ == modify **then**
create new node $w^*_{i_j} = (r_{i_j}, h^*_{i_j})$, and replace w_{i_j} with $w^*_{i_j}$ in T; {note that $w_{i_j} = (r_{i_j}, h_{i_j})$}
else if $U[j]$ == insert **then**
create new node $w^*_{i_j} = (1, h^*_{i_j})$, $w'_{i_j} = (r_{i_j} + 1, h'_{i_j})$, where $h'_{i_j} = H(r_{i_j} + 1 \| h^*_{i_j} \| h_{i_j})$, and replace w_{i_j} in T with w'_{i_j} which has left child $w^*_{i_j}$ and right child w_{i_j};
else {$U[j]$ == delete}
if sn is a leaf node **then**
delete w_{i_j} from T, and replace pn with sn;
else {sn is a non-leaf node}
delete w_{i_j} from T, and replace pn with T_{sn};
{where sn, pn are the sibling node, parent node of w_{i_j} respectively, and T_{sn} is a subtree rooted at sn}
end if
end if
end for
return T' and $W = (W[1], W[2], \cdots, W[c])$; {note that if $U[j]$ == modify then $W[j] = w^*_{i_j}$, else if $U[j]$ == insert then $W[j] = w'_{i_j}$, else $U[j]$ == delete then $W[j]$ = NULL}

When multiple update operations are requested, a straightforward way is performing these updates one by one. This method, however, incurs additional computation overhead at the auditor side, as he computes some repetitive hash values. To see more clearly, let us take the Fig. 2 for example (page 4). Given two update commands $(3, \text{modify}, h^*_3)$, $(7, \text{delete}, \text{NULL})$ received from client, if auditor first modifies the node w_3, then he needs to calculate hashes of $w_{16}, w_{21}, w_{25}, root$. Later, when auditor deletes the node w_7, he needs to calculate hashes of $w_{22}, w_{25}, root$. There are 5 different nodes that auditor needs to recalculate hashes, but he does 7 hash calculations. This situation will be aggravated with the number of update operations increases. In addition, when multiple updates are verified individually, it also wastes additional bandwidth and computation resources at both CSP and client side, as analyzed in previous strawman solution (cf. Section 3.1).

To conquer the above problems, we design an Algorithm 3 based on MLA solution to handle multiple updates in a batch way. This algorithm is triggered by client, who first computes all the hash and tag values of the new data blocks and then sends the update command uc to auditor and CSP (as shown in lines 1–8). After receiving uc, auditor updates the tree T and sends the update proof up to CSP (as shown in lines 9–13). Then C-

Algorithm 5 UpdateMultiProof$(U, W, n, C, \sqcup_p)$

Input: the update operation vector U, the new node vector W, the original total number of data blocks n, challenge vector C and multi-proof $\sqcup_p$.
Output: the updated total number of data blocks n^*, challenge vector C^* and multi-proof $\sqcup_p^*$.

```
for j = c to 1 do
  if U[j] == modify then
    update ⊔p by replacing the j-th E-marked element (E, w_{i_j}) with (E, w*_{i_j});
  else if U[j] == insert then
    n = n + 1; C = C(j-after) + 1;
    update ⊔p by replacing the j-th E-marked element (E, w_{i_j}) with (E, w'_{i_j});
  else {U[j] == delete}
    n = n − 1; C = C(j-after) − 1;
    {let (d, v) denote the first element below the j-th E-marked element in ⊔p, where v = (r, h)}
    if d == E then
      delete the j-th element C[j] from C;
      update ⊔p by deleting the j-th E-marked element (E, w_{i_j}) and its corresponding I-marked element;
    else if d == I then
      delete the j-th element C[j] from C;
      update ⊔p by deleting the j-th E-marked element (E, w_{i_j}) and deleting the element (I, v);
    else if d == R then
      C = C(j) − r;
      update ⊔p by deleting the j-th E-marked element (E, w_{i_j}) and replacing (R, v) with (E, v);
    else {d == L}
      update ⊔p by deleting the j-th E-marked element (E, w_{i_j}) and replacing (L, v) with (E, v);
    end if
  end if
end for
the updated n, C and ⊔p are denoted as n*, C* and ⊔p*, respectively;
return n*, C* and ⊔p*;
```

SP verifies the correctness of up and later client also checks it (as shown in lines 14–15). If one of the verifications fails, this algorithm returns False; otherwise client sends update information ui to CSP. After receiving ui, CSP updates the processed data (as shown in lines 16–24). At last, the three participants sign the updated public parameter $\mathbb{P}^* = \{n^*, h_{root^*}\}$, and output True (as shown in lines 25–26). Note that to save space, we omit the signature verification in each above interaction.

In Algorithm 6, our method to verify the update proof up consists of four parts. First, the multi-proof $\sqcup_p$ is verified according to n, h_{root}, C. Second, based on uc and the verified $\sqcup_p$, we construct the new node vector W by the similar way in Algorithm 4. Third, by invoking Algorithm 5 UpdateMultiProof$(U, W, n, C, \sqcup_p)$, the updated values $n^*, C^*, \sqcup_p^*$ are obtained. Fourth, the new Merkle root h_{root^*} sent by auditor is verified according to the correct values $n^*, C^*, \sqcup_p^*$. If all the above verifications pass, we confirm the auditor did perform updates correctly; otherwise we reject the update proof up.

Algorithm 6 VerifyUpdateProof(up, uc, n, h_{root})

Input: the update proof up, the update command uc, the total number of data blocks n and the Merkle root h_{root}. {where $up = (\sqcup_p, h_{root^*})$, $uc = (C, U, H^*)$}
Output: True or False.

```
if VerMultiProof (⊔p, n, h_root, C) == Accept then
  based on uc and ⊔p obtain the new node vector W in a way similar to the auditor in Algorithm 4;
  obtain n*, C* and ⊔p* by calling Algorithm 5;
  if VerMultiProof (⊔p*, n*, h_root*, C*) == Accept then
    return True;
  else {VerMultiProof (⊔p*, n*, h_root*, C*) == Reject}
    return False;
  end if
else {VerMultiProof (⊔p, n, h_root, C) == Reject}
  return False;
end if
```

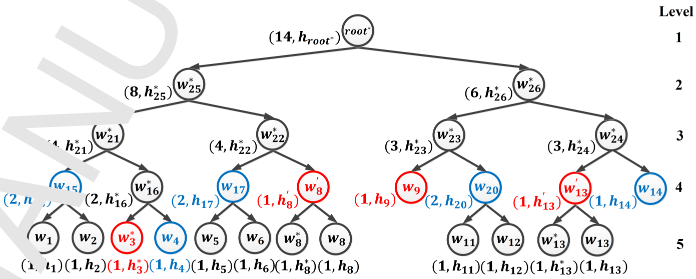


Fig. 4: The final tree T^* after updates, where the updated challenge nodes are marked with red, the necessary nodes needed for verification are marked with blue.

In the example of Fig. 2 (page 4), if the update command $uc = (C, U, H^*)$ is performed by auditor correctly, where $C = (3, 7, 8, 10, 13)$, $U =$ (modify, delete, insert, delete, insert), $H^* = (h_3^*, \text{NULL}, h_8^*, \text{NULL}, h_{13}^*)$, then the final tree T^* whose updated nodes are marked with $*$ is shown in Fig. 4. In addition, the updated multi-proof $\sqcup_p^*$ corresponding to $C^* = (3, 7, 9, 12)$ is listed in Fig. 3(b) (page 5).

3.2.6. ImpartialArbitration Protocol

This protocol will be triggered if there are any disputes among the involved three participants. In this case, the honest participant resorts to a trusted third party, called arbitrator, to vindicate his/her innocence and detect the malicious behaviors. As analyzed in Section 2.2, if a scheme is secure against two malicious participants, then it naturally is secure against any one of them. Therefore, it suffices to take into account the following three cases where only one participant is honest.

Case 1: Only client is honest. If data corruption has occurred in CSP, then client can detect this malicious incident even CSP colludes with auditor. The reason is that the proof of appointed logs $\rho^{(B)}$ generated by them can not pass the client's checking.

If this evidence is submitted to arbitrator, then the collusion of CSP and auditor will be catched.

Case 2: Only CSP is honest. If auditor colludes with client, and claims that at least one of the data blocks m_{i_η} $(1 \leq \eta \leq l)$ is lost or corrupted by CSP. In this situation, CSP only needs to submit all these blocks to arbitrator. Based on h_{root} confirmed by three participants, arbitrator first authenticates the hash values h_{i_η} $(1 \leq \eta \leq l)$ received from auditor. If this verification is passed, then arbitrator computes $H_2(m_{i_\eta})$ and compares it with the authenticated h_{i_η}. If $H_2(m_{i_\eta}) = h_{i_\eta}$ holds for $1 \leq \eta \leq l$, then arbitrator can prove that all these data blocks are really stored at CSP.

Case 3: Only auditor is honest. If client deliberately claims that the auditor does not do his auditing work honestly, then the log file allows auditor to prove his innocence even client colludes with CSP. This is because the log file reflects the auditor's past auditing work, arbitrator can check log entry $L^{(b)}$ to prove auditor's behavior at the corresponding point in time. This checking involves Merkle root h_{root}, CSP's signature and client's public key, so they do not deny the verification result. If the verification passes, auditor is honest without doubt.

3.2.7. Discussion and Extension

For notational simplicity, we present the ODPDP scheme in the symmetric bilinear group, where the two parameters of the pairing belong to the same group G. In fact, our scheme can also be constructed based on an asymmetric bilinear map. In what follows, we show how to extend our scheme to an asymmetric setting. In general, the asymmetric bilinear map has the form $e : G_1 \times G_2 \rightarrow G_T$, where G_1, G_2 and G_T are three multiplicative cyclic groups of prime order q. Let g and g' be the generators of G_1 and G_2, respectively. The difference is that now the public keys λ, $g_1 = g^{\alpha_1}, \cdots, g_s = g^{\alpha_s}$ live in G_1, and the public key $y = (g')^x$ lives in G_2. Accordingly, the corresponding notations involved in this extension should be adjusted to cater to the asymmetric setting. As shown in [7, 16], the symmetric version has a simpler expression, while the asymmetric version has faster operations under the same security level.

4. Security Analysis

The correctness of our scheme has been demonstrated at the end of **AuditData** and **AuditLog** protocols, thus is omitted here. In this section, we prove the security of our ODPDP scheme with respect to three properties: authenticity, liability and extractability.

Theorem 1. (Authenticity) *Assume that hash function H_1 is collision resistant, RBMT guarantees the integrity of hash values stored in its leaf nodes.*

Proof. First, we show that the malicious auditor is not able to deceive client and pass her verification by replacing a challenged leaf node with another leaf node. The reason is that the first property of Algorithm 2 can defend against this replacing-attack. Second, it is impossible for the auditor to forge a valid proof of the challenged leaf nodes if one of these nodes has been tampered. If auditor can make client accept a forged proof of the challenged leaf nodes, then we can break the collision resistance of hash function H_1 by using a simple reduction as in [27]. The reduction keeps a local copy of RBMT, if auditor forges a valid proof, then the reduction would output the proof together with an authentic proof in the local copy as a collision, which contradicts our initial assumption.

Summing up, we prove that RBMT can protect the integrity of hash values stored in its leaf nodes as long as the used hash function is collision resistant. □

Theorem 2. (Liability) *In our scheme, the honest auditor can attest any party that he did his auditing work correctly in case of conflicts while the malicious auditor will fail.*

Proof. We first analyze the case where the auditor is honest while the other two participants are malicious. Notice that the auditor's work consists of three parts, namely, generating the challenge $Q^{(b)}$, computing the value $h^{(b)}$ and verifying the Eq. (2). Due to the fact that any challenge generated from Bitcoin can be reconstructed later on, arbitrator can check if the challenge $Q^{(b)}$ is correct. Based on the authenticated hash values, the arbitrator can check if the value $h^{(b)}$ is correct. The verification of Eq. (2) can be redone by the arbitrator with the public key pk. So the honest auditor can be protected by his log file, which is an objective evidence that can be used to prove the auditor's well behavior.

On the other hand, if auditor is malicious and irresponsible to his auditing work, then his misbehavior would be detected by client during **AuditLog** protocol. This is because the client can audit the log file to check whether the auditor was honest to do the past auditing work whenever she wants. Thus, the malicious auditor can not prove he is well behaved unless he did his auditing work correctly in the past. □

The extractability of our scheme is based on computational Diffie-Hellman (CDH) assumption in bilinear group G, which is defined as follows.

Definition 3. (CDH Assumption) *For any probabilistic polynomial time adversary $\mathcal{A}$, the advantage of the adversary on computing u^a given $g, g^a, u \in G$ is negligible, namely,*

$$\Pr[\mathcal{A}(g, g^a, u \in G) \rightarrow u^a \in G : \forall a \in \mathbb{Z}_q^*] \leq \epsilon.$$

Theorem 3. (Extractability) *Assuming the CDH assumption holds in bilinear group G, for any probabilistic polynomial time adversary $\mathcal{A}$ who has corrupted CSP and auditor (or auditor and client), if $\mathcal{A}$ forges a valid proof successfully, then there exists an extraction algorithm that can recover the challenged data blocks from $\mathcal{A}$—except possibly with negligible probability.*

Proof. We discuss the first scenario where the client is honest while the other participants are corrupted by $\mathcal{A}$. Assume that there exists an adversary $\mathcal{A}$ who wins the following game on a challenge Q launched by the challenger, we show how to construct a simulator that is able to extract the challenged data blocks.

For the CDH problem, the simulator is given values $g, g^a, u \in G$, its goal is to find a value u^a. The simulator plays the part of the game challenger, and simulates an ODPDP environment for $\mathcal{A}$ with the following differences.

In setup phase, the simulator sets the value y to g^a, which means that it does not know the secret value a.

In query phase, the adversary $\mathcal{A}$ queries the store oracle adaptively, the simulator answers $\mathcal{A}$'s store queries as below.

For each z $(1 \leq z \leq s)$, the simulator first selects random values $b_z, c_z \in \mathbb{Z}_q$ and sets $g_z = g^{b_z} \cdot u^{c_z}$. Then the simulator selects a random value $d \in \mathbb{Z}_q^*$ and sets $\lambda = u^d$.

Upon receiving a hash query for a block $m_i = m_{i1}\|m_{i2}\|\cdots\|m_{is}$ $(1 \leq i \leq n)$, the simulator computes $H_2(m_i) = -d^{-1} \cdot \sum_{z=1}^{s} c_z m_{iz}$, and sends it back to $\mathcal{A}$.

Upon receiving a tag query for a block m_i $(1 \leq i \leq n)$, the simulator computes $\sigma_i = (g^a)^{\sum_{z=1}^{s} b_z m_{iz}}$, and responds $\mathcal{A}$ with it. Note that the tag σ_i generated by the simulator is in accord with the protocol specification, since the following equation holds

$$\begin{aligned}\sigma_i &= (\lambda^{H_2(m_i)} \cdot g^{\sum_{z=1}^{s} \alpha_z m_{iz}})^a \\ &= ((u^d)^{-d^{-1} \cdot \sum_{z=1}^{s} c_z m_{iz}} \cdot \prod_{z=1}^{s} g_z^{m_{iz}})^a \\ &= (u^{-\sum_{z=1}^{s} c_z m_{iz}} \cdot \prod_{z=1}^{s} (g^{b_z} \cdot u^{c_z})^{m_{iz}})^a \\ &= (u^{-\sum_{z=1}^{s} c_z m_{iz}} \cdot g^{\sum_{z=1}^{s} b_z m_{iz}} \cdot u^{\sum_{z=1}^{s} c_z m_{iz}})^a \\ &= (g^a)^{\sum_{z=1}^{s} b_z m_{iz}}.\end{aligned}$$

If $\mathcal{A}$ can forge a valid proof of possession for the data blocks determined by the simulator's challenge Q, then the simulator can use $\mathcal{A}$ to solve the CDH problem. Let a correctly computed proof be $\rho = \{\mu_1, \mu_2, \cdots, \mu_s, \sigma\}$, that satisfies the verification equation by the correctness of our scheme

$$e(\sigma, g) = e(h \cdot \prod_{z=1}^{s} g_z^{\mu_z}, y)$$

Let $\rho' = \{\mu'_1, \mu'_2, \cdots, \mu'_s, \sigma'\}$ be a valid proof forged by $\mathcal{A}$, so this proof also satisfies the verification equation

$$e(\sigma', g) = e(h \cdot \prod_{z=1}^{s} g_z^{\mu'_z}, y)$$

If $\mu'_z = \mu_z$ for all $1 \leq z \leq s$, then the simulator has already successfully obtained the correct μ_z $(1 \leq z \leq s)$. We analyze the opposite case where at least one of $\mu'_z = \mu_z$ $(1 \leq z \leq s)$ does not hold. Hence, it follows from the verification equation that $\sigma \neq \sigma'$. Now, dividing the verification equation for the forged proof by the verification equation for the correct proof, we obtain

$$\begin{aligned}e(\sigma' \cdot \sigma^{-1}, g) &= e(\prod_{z=1}^{s} g_z^{\Delta\mu_z}, y) \\ &= e(\prod_{z=1}^{s} (g^{b_z} \cdot u^{c_z})^{a\Delta\mu_z}, g),\end{aligned}$$

where $\Delta\mu_z = \mu'_z - \mu_z$ for each $1 \leq z \leq s$. We can further obtain

$$\begin{aligned}\sigma' \cdot \sigma^{-1} &= \prod_{z=1}^{s} (g^{b_z} \cdot u^{c_z})^{a\Delta\mu_z} \\ &= (g^a)^{\sum_{z=1}^{s} b_z \Delta\mu_z} \cdot (u^a)^{\sum_{z=1}^{s} c_z \Delta\mu_z}.\end{aligned}$$

So far, we have found a solution to the CHD problem

$$u^a = (\sigma' \cdot \sigma^{-1} \cdot y^{-\sum_{z=1}^{s} b_z \Delta\mu_z})^{\frac{1}{\sum_{z=1}^{s} c_z \Delta\mu_z}},$$

unless the denominator $\sum_{z=1}^{s} c_z \Delta\mu_z$ is zero. However, note that at least one of $\Delta\mu_z$ $(1 \leq z \leq s)$ is nonzero, and the values c_z $(1 \leq z \leq s)$ are information-theoretically hidden from the adversary $\mathcal{A}$, so the denominator is zero only with a negligible probability $1/q$.

For each sum of the form $\mu_z = a_1 m_{i_1 z} + a_2 m_{i_2 z} + \cdots + a_l m_{i_l z}$, we show that the simulator may extract the sectors $m_{i_1 z}, m_{i_2 z}, \cdots, m_{i_l z}$ in polynomially-many interactions with the adversary $\mathcal{A}$. By running **AuditLog** protocol repeatedly, the simulator may obtain l independent linear equations in the variables $m_{i_1 z}, m_{i_2 z}, \cdots, m_{i_l z}$. Then the simulator solves these equations to get the sectors $m_{i_1 z}, m_{i_2 z}, \cdots, m_{i_l z}$. Therefore, the simulator can extract the data blocks $m_{i_1}, m_{i_2}, \cdots, m_{i_l}$ dictated by the challenge Q. This concludes the first scenario.

It remains to discuss the next scenario, where CSP is honest while the others are malicious. The simulator takes as input the data M stored at the honest CSP side, and can trivially extract the challenged data blocks. □

In conclusion, according to Definition 2 (cf. Section 2.2), we have argued that our scheme is secure based on the above three theorems.

Note that the auditor can obtain the challenged data blocks from sufficiently many correct proofs. In other words, the proofs returned by CSP may leak the data content to the auditor during the auditing process. As mentioned in Section 2.2, we do not consider the confidentiality of the data, which is beyond the scope of the problem we study here. If the confidentiality of the data needs to be protected, there are two common solutions to be adopted. One is to incorporate privacy-preserving techniques [18–21] into integrity auditing scheme to prevent the auditor from learning knowledge about the data, which is one of the important research directions in this area; the other is relatively straightforward: client encrypts her data prior to starting the integrity auditing scheme.

5. Performance Analysis

5.1. Theoretical Analysis

The features of our ODPDP scheme are listed in Table 4. We also include a comparison of related schemes [9–11, 15–18], in which the scheme [18] refers to the dynamic version. Note that the Setup and Store protocols are taken as a protocol in some schemes, so we will put them together for ease of comparison. Now, we explain some notations used in this table.

Table 4: Theoretical comparison of property, computation cost and communication cost.

Scheme	Property		Computation cost			Communication cost	
	Audit type	Update type	Setup and Store	AuditData	AuditLog	AuditData	AuditLog
DPDP[9]	private	single	$n\text{Exp}_{\mathbb{Z}_N^*}$	$(l+1)\text{Exp}_{\mathbb{Z}_N^*}$	N/A	$O(\log n)$	N/A
FlexDPDP[10]	private	batch	$n\text{Exp}_{\mathbb{Z}_N^*}$	$(l+1)\text{Exp}_{\mathbb{Z}_N^*}$	N/A	$O(\log \frac{n}{l})$	N/A
Scheme[11]	public	single	$2(ns+1)\text{Exp}_G$	$(2l+1)\text{Exp}_G$ +4Pair	N/A	$O(\log ns)$	N/A
IPIC-DG[17]	public	N/A	$(ns+4n+10)\text{Exp}_G$+$(3n+4)$Pair+$(3n+2)\text{Exp}_{G_T}$	$(10l+s)\text{Exp}_G$+ $5l$Pair+$4l\text{Exp}_{G_T}$	N/A	$O(1)$	N/A
SEPDP[18]	public	single	$(s+1)\text{Exp}_G$	$(s+3)\text{Exp}_G$	N/A	$O(1)$	N/A
Fortress[15]	outsourced	N/A	$(ns+10n+4s)\text{Exp}_{\mathbb{Z}_N^*}$	0	0	$O(1)$	$O(1)$
DOA[16]	outsourced	single	$(n+1)\text{Exp}_{\mathbb{Z}_N^*}$	$(l+5)\text{Exp}_{\mathbb{Z}_N^*}$	$5\text{Exp}_{\mathbb{Z}_N^*}$	$O(1)$	$O(\log \frac{n}{c})$
ODPDP(ours)	outsourced	batch	$(2n+s+1)\text{Exp}_G$	$(l+s+1)\text{Exp}_G$ +2Pair	3Exp_G	$O(1)$	$O(\log \frac{n}{c})$

First, let Pair denote the pairing operation on the group G, and let Exp_G, Exp_{G_T}, $\text{Exp}_{\mathbb{Z}_N^*}$ denote the exponentiation operations on the groups G, G_T, $\mathbb{Z}_N^*$ respectively, where N is RSA modulus. Second, let n denote the total number of data blocks and s denote the number of sectors per block. Third, let l and c denote the number of challenged data blocks in **AuditData** and **AuditLog** protocols, respectively. Finally, let N/A denote not applicable in that case, e.g., the scheme can not support dynamic update or outsourced auditing.

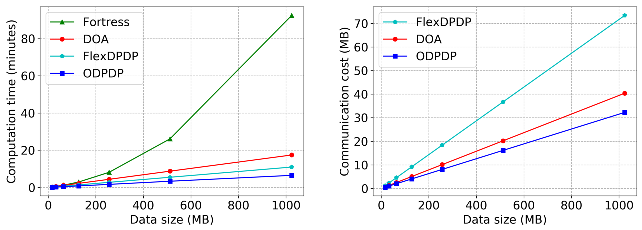
We can see from Table 4 that only Fortress [15], DOA [16] and our ODPDP support outsourced auditing property, and only FlexDPDP [10] and our ODPDP support batch update property. For computation cost, we only make the comparison with regards to exponentiation and pairing operations, since the other operations such as multiplication and addition are relatively lightweight. IPIC-DG [17] adopts the ZKP protocol to achieve the anonymity of users, which incurs more exponentiation operations than the others. The results show that SEPDP [18] requires less exponentiations than the others. Observe that only DOA [16] and our scheme can support outsourced auditing and dynamic update simultaneously, so we mainly focus on the two schemes to make a fair comparison. Note that only the number of cryptographic operations may not reflect the computation performance of the two schemes, since they are based on different group setting. For a more realistic comparison, please refer to the next Section 5.2. In addition, we find that our scheme exhibits the same communication performance with DOA [16] for the **AuditData** and **AuditLog** protocols. Moreover, our scheme reduces the amortized proof per update from $1+\log n$ to $1+\log(n/c)$ due to the proposed batch update algorithm, where c denotes the number of updated data blocks.

5.2. Experiment Evaluation

In this section, we evaluate the performance of our ODPDP scheme in terms of computation time and communication cost. As a basis for comparison, we implemented the prototype of our ODPDP scheme and the most related schemes (FlexDPDP [10], Fortress [15] and DOA [16]) in Python. All cryptographic operations were conducted in Charm [28, 29], a framework of rapidly prototyping cryptosystems. In our implementation, we relied on the Python built-in random number generator, the secure hash algorithm H_1 was instantiated by using SHA256, and the utilized elliptic curve was MNT224. In addition, each random challenge was extracted by inputting a Bitcoin block hash, which can be obtained by using a *getblockhash* tool as in [15]. We deployed our experiments on a machine possessing 16-core Intel Xeon E5-2620 v4 CPU @ 2.10GHz, 32GB of RAM, and 20MB of L3 cache, running CentOS Linux release 7.5.1804 (Linux kernel 3.10.0-514.el7.x86_64).

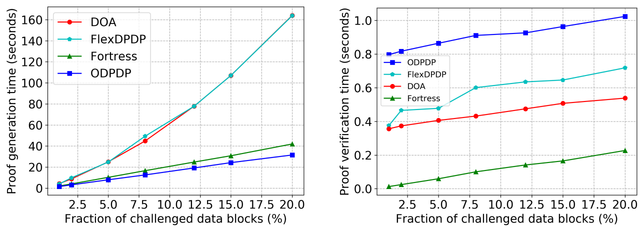
For ease of comparison, we used the same scenario as in previous work [16], where the size of outsourced data was set to 1 GB for testing. As suggested in [15, 16], the block size was set to 16 KB for the most balanced performance. In addition, the sector size per data block in ODPDP and Fortress was set to 223 bits, and the size of RSA modulus in FlexDPDP, Fortress and DOA was set to 2048 bits. All evaluation results are the average of 10 runs.

Pre-processing performance: Fig. 5(a) shows the pre-processing time (i.e., **Store** protocol) as a function of data size for all investigated schemes. To optimize store protocol in Fortress, we leverage the fact that auditor's tag is homomorphic and verify all the tags in a single batch way. The results show that the computation time required to pre-process 1GB outsourced data in ODPDP is almost 1.7, 2.7 and 14 times faster than FlexDPDP, DOA and Fortress, respectively. This does mirror the advantage of our EHVT, which can reduce store effort from $O(s)$ exponentiations to $O(1)$ exponentiations. The communication cost incurred by authenticated data structure during **Store** protocol under different data sizes is illustrated in Fig. 5(b). In order to generate auditor's tags, Fortress must download the client's raw data from CSP to auditor, so it's communication cost is naturally larger than FlexDPDP, DOA and ODPDP, thus is omitted here. Our findings show that the communication cost of FlexDPDP is larger than the other schemes, since the former stores 2048 bits tag value in each leaf node while the others only need to store relatively small hash value. We also find that the communication cost of ODPDP is smaller than that of DOA, because the latter need to store additional in-



(a) (b)

Fig. 5: (a) Computation time for client when pre-processing the outsourced data w.r.t. data size; (b) Communication cost incurred by authenticated data structure during **Store** protocol w.r.t. data size.



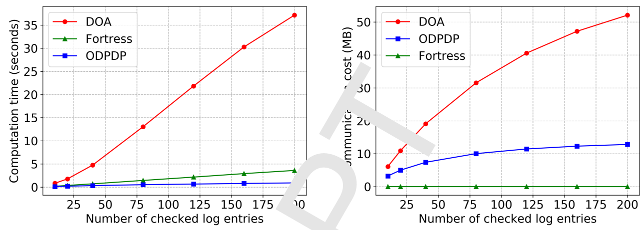
(a) (b)

Fig. 6: (a) Proof generation time of **AuditData** protocol w.r.t. the fraction of challenged data blocks; (b) Proof verification time of **AuditData** protocol w.r.t. the fraction of challenged data blocks.

formation (i.e., status value and height value) for each node in Merkle tree.

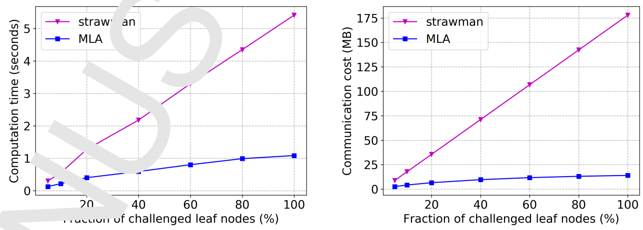
AuditData performance: The time incurred by generating proof at CSP side once w.r.t. the fraction of challenged data blocks is shown in Fig. 6(a). Our results show that the proof generation time increases almost linearly with the fraction of challenged data blocks for all investigated schemes. We can see that the computation time required by DOA is very close to that of FlexDPDP, and both are larger than the other two schemes. The reason is that CSP's computation time in DOA and FlexDPDP is dominated by computing the aggregated tag (involving exponentiation operations) which consumes considerable computation effort compared with the other schemes. In Fig. 6(b), we measure the computation time incurred by verifying proof w.r.t. the fraction of challenged data blocks. It can be observed that the computation time in Fortress is smaller than the other three schemes, this is because verification in Fortress only involves addition and multiplication operations while the other schemes require exponentiation operation. Due to only the constant number of exponentiations is required in DOA and FlexDPDP while ODPDP needs to perform s exponentiations, so ODPDP is slightly slower than DOA and FlexDPDP in terms of proof verification time.

AuditLog performance: In this experiment, the number of challenged data blocks was set to 460 to achieve 99% probability of misbehavior detection, as suggested in [6]. Fig. 7(a) presents the computation time incurred by client to audit the auditor's past work once under the number of checked log entries. As expected, the time to check the log file almost increases lin-



(a) (b)

Fig. 7: (a) Computation time for client to check auditor's log once w.r.t. the number of checked log entries; (b) Communication cost during **AuditLog** protocol w.r.t. the number of checked log entries.



(a) (b)

Fig. 8: Performance of MLA solution in comparison to the strawman solution w.r.t. the fraction of challenged leaf nodes. (a) Computation time compared; (b) Communication cost compared.

early with the number of appointed log entries in all investigated schemes. But ODPDP exhibits a better log audit performance at client side when compared to the other two schemes. In Fig. 7(b), we evaluate the communication cost during **AuditLog** protocol w.r.t. the number of verified log entries. The different prices of dynamic update caused by ODPDP and DOA are respectively presented by the distances between the line labeled Fortress and other lines, which come from the Merkle tree proof. The results show that ODPDP results in a smaller communication cost than DOA, as the node size of Merkle tree in DOA is larger than our scheme. Note that FlexDPDP is not applicable for this evaluation, since it does not provide the log audit mechanism for the client.

Comparison with the strawman solution: In an additional experiment, we evaluate the improved performance of MLA solution against the straightforward solution, which can only authenticate the challenged leaf nodes one by one. As shown in Fig. 8(a) and Fig. 8(b), it can be observed that our solution considerably improves the performance over the traditional solution in terms of both computation time and communication cost. The reason is that the multi-proof of challenged leaf nodes can be generated and verified without the repetitive calculations and transmissions. With the number of challenged leaf nodes increases, the distance between the two lines will become more pronounced. This evaluation suggests that our MLA solution is more suitable for ODPDP scheme.

6. Related Work

Data integrity auditing is an important technique for secure cloud storage, which has been extensively investigated in the past few years. The first proofs of retrievability (POR) scheme was presented by Juels and Kaliski [5], which relies on sentinel blocks hidden among actual data blocks to detect data corruption by the untrusted server. But the number of challenges launched by client is limited, because the sentinel blocks might be used up after some challenge-response interactions. At the same year, the first provable data possession (PDP) scheme was proposed by Ateniese et al. [6], which is based on homomorphic verifiable tag (HVT) and random sampling. The scheme supports an unbounded number of challenges compared with the one in [5], and can be modified to offer public verifiability. After that, two compact POR schemes were presented by Shacham and Waters [7], which provide full security proofs. The first scheme with private verifiability is built on pseudorandom function, and the second one with public verifiability is built from BLS signature [23].

Unfortunately, the aforementioned schemes only apply to static scenario, where the outsourced data is never changed by client. To cope with dynamic scenario, Ateniese et al. [8] proposed a scalable and efficient PDP scheme, which only allows partial update operations while block insertion at any position is not supported. To support full update operations, Erway et al. [9] presented a dynamic PDP scheme based on skip list. Esiner et al. [10] extended this work to support variable block-size update by using FlexList.

To relieve clients from the burden of frequent auditing, Wang et al. [11] proposed a public auditing scheme that relies on trusted TPA to check data integrity. The scheme also supports full dynamic updates by using Merkle tree. To support fine-grained update, Liu et al. [12] presented a dynamic public auditing scheme by employing variable-size block based Merkle tree. By migrating version information of clients' data from CSP to TPA, a dynamic-hash-table based public auditing scheme was proposed by Tian et al. [13] in order to reduce computation overhead and communication cost. All existing public auditing schemes, however, rely on an assumption that TPA is trusted and performs auditing task honestly on behalf of client.

To provide security guarantees that have not been covered in previous auditing schemes, an outsourced auditing scheme was first presented by Armknecht et al. [15], which can protect against any one malicious participant and against collusion of any two participants. But their scheme just applies to static data, which can not cater to dynamic scenario where clients expect to update their outsourced data. Rao et al. [16] extended it to support dynamic update, but it only can handle multiple update operations one by one, which may limit update efficiency in practice and is also one of main focuses in our paper.

7. Conclusion

Outsourced auditing is a novel business model for cloud storage service, where a financial contract is established between clients and auditor by which clients can be assured that the integrity of their data is continually monitored by auditor. In this paper, we proposed an ODPDP scheme that can resist any dishonest participant and collusion, which is not satisfied by traditional auditing schemes. Due to the proposed EHVT, our scheme can considerably enhance computation efficiency with respect to exponentiation operation, especially for clients during **Store** and **AuditLog** protocols. In addition, our scheme can not only authenticate multiple leaf nodes and their indices all together but also handle multiple updates at once. Note that both the tag construction and the batch processing are tailor-made for ODPDP scheme. Security analysis and experiments showed that our scheme is provably secure and highly efficient. In terms of future work, we plan to explore effective solutions to incorporate the privacy-preserving technique into the ODPDP scheme.

Acknowledgment

This work was supported by NSFC (Grant No. 61502044).

References

[1] P. Mell, T. Grance, The NIST definition of cloud computing, in: NIST Special Publication 800-145, 2011.

[2] S. Kamara, K. Lauter, Cryptographic cloud storage, in: FC Workshops, 2010, pp. 136–149.

[3] M. Sookhak, H. Talebian, E. Ahmed, A. Gani, M.K. Khan, A review on remote data auditing in single cloud server: taxonomy and open issues, J. Netw. Comput. Appl. 43 (2014) 121–141.

[4] M. Kolhar, M.M. Abu-Alhaj, S.M.A. El-atty, Cloud data auditing techniques with a focus on privacy and security, IEEE Secur. Priv. 15 (1) (2017) 42–51.

[5] A. Juels, B.S. Kaliski Jr, PORs: Proofs of retrievability for large files, in: ACM CCS, 2007, pp. 584–597.

[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song, Provable data possession at untrusted stores, in: ACM CCS, 2007, pp. 598–610.

[7] H. Shacham, B. Waters, Compact proofs of retrievability, in: ASIACRYPT, 2008, pp. 90–107.

[8] G. Ateniese, R.D. Pietro, L.V. Mancini, G. Tsudik, Scalable and efficient provable data possession, in: SecureComm, 2008, pp. 1–10.

[9] C.C. Erway, A. Küpçü, C. Papamanthou, R. Tamassia, Dynamic provable data possession, in: ACM CCS, 2009, pp. 213–222.

[10] E. Esiner, A. Kachkeev, S. Braunfeld, A. Küpçü, O. Ozkasap, FlexDPDP: Flexlist-based optimized dynamic provable data possession, ACM Trans. Storage 12 (4) (2016) 1–44.

[11] Q. Wang, C. Wang, K. Ren, W. Lou, J. Li, Enabling public auditability and data dynamics for storage security in cloud computing, IEEE Trans. Parall. Distr. 22 (5) (2011) 847–859.

[12] C. Liu, J. Chen, L.T. Yang, X. Zhang, C. Yang, R. Ranjan, R. Kotagiri, Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates, IEEE Trans. Parall. Distr. 25 (9) (2014) 2234–2244.

[13] H. Tian, Y. Chen, C.C. Chang, H. Jiang, Y. Huang, Y. Chen, J. Liu, Dynamic-hash-table based public auditing for secure cloud storage, IEEE Trans. Serv. Comput. 10 (5) (2017) 701–714.

[14] T. Tu, L. Rao, H. Zhang, Q. Wen, J. Xiao, Privacy-preserving outsourced auditing scheme for dynamic data storage in cloud, Secur. Commun. Netw. (2017) 1–17.

[15] F. Armknecht, J.M. Bohli, G.O. Karame, Z. Liu, C.A. Reuter, Outsourced proofs of retrievability, in: ACM CCS, 2014, pp. 831–843.

[16] L. Rao, H. Zhang, T. Tu, Dynamic outsourced auditing services for cloud storage based on batch-leaves-authenticated Merkle hash tree, IEEE Trans. Serv. Comput. doi:10.1109/TSC.2017.2708116.

[17] A. Hu, R. Jiang, B. Bhargava, Identity-preserving public integrity checking with dynamic groups for cloud storage, IEEE Trans. Serv. Comput. doi:10.1109/TSC.2018.2859999.

[18] S.K. Nayak, S. Tripathy, SEPDP: Secure and efficient privacy preserving provable data possession in cloud storage, IEEE Trans. Serv. Comput. doi:10.1109/TSC.2018.2820713.

[19] C. Wang, Q. Wang, K. Ren, W. Lou, Privacy-preserving public auditing for data storage security in cloud computing, in: IEEE INFOCOM, 2010, pp. 1–9.

[20] K. Yang, X. Jia, An efficient and secure dynamic auditing protocol for data storage in cloud computing, IEEE Trans. Parall. Distr. 24 (9) (2013) 1717–1726.

[21] J. Li, L. Zhang, J.K. Liu, H. Qian, Z. Dong, Privacy-preserving public auditing protocol for low performance end devices in cloud, IEEE Trans. Inf. Foren. Sec. 11 (11) (2016) 2572–2583.

[22] J. Garay, A. Kiayias, N. Leonardos, The Bitcoin backbone protocol: analysis and applications, in: EUROCRYPT, 2015, pp. 281–310.

[23] D. Boneh, B. Lynn, H. Shacham, Short signatures from the Weil pairing, in: ASIACRYPT, 2001, pp. 514–532.

[24] R.C. Merkle, A certified digital signature, in: CRYPTO, 1990, pp. 218–238.

[25] Bitcoin real-time stats and tools, http://blockexplorer.com/q, (accessed 17 July 2018).

[26] Y. Rouselakis, B. Waters, Efficient statically-secure large-universe multi-authority attribute-based encryption, in: FC, 2015, pp. 315–332.

[27] C. Papamanthou, R. Tamassia, Time and space efficient algorithms for two-party authenticated data structures, in: ICICS, 2007, pp. 1–15.

[28] Charm, https://pypi.python.org/pypi/charm-crypto/0.43, (accessed 13 August 2018).

[29] J.A. Akinyele, C. Garman, I. Miers, M.W. Pagano, M. Rushanan, M. Green, A.D. Rubin, Charm: A framework for rapidly prototyping cryptosystems, J. Crypto. Eng. 3 (2) (2013) 111–128.

Author Biography

Wei Guo received M.S. degree in applied mathematics from Shaanxi Normal University in 2016. Now, he is a Ph.D. candidate in Beijing University of Posts and Telecommunications. His research interests focus on cloud computing security and applied cryptography.

Hua Zhang received B.S. and M.S. degree from Xidian University in 2002 and 2005, respectively, and Ph.D. degree in cryptography from Beijing University of Posts and Telecommunications (BUPT) in 2008. Now, she is an associate professor in Institute of Network Technology, BUPT. Her research interests include cryptography and information security.

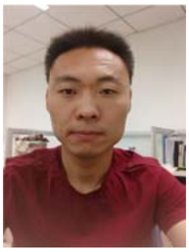
Sujuan Qin received Ph.D. degree in cryptography from Beijing University of Posts and Telecommunications (BUPT) in 2008. Now, she is an associate professor in Institute of Network Technology, BUPT. Her research interests include cryptography, information security, and quantum cryptography.

Fei Gao received Ph.D. degree in cryptography from Beijing University of Posts and Telecommunications (BUPT) in 2007. Now, he is a professor in Institute of Network Technology, BUPT. His research interests include cryptography, information security, and quantum algorithm. Prof. Gao is a member of the Chinese Association for Cryptologic Research.

Zhengping Jin received B.S. and M.S. degrees from Anhui Normal University in 2004 and in 2007, respectively, and Ph.D. degree in cryptography from Beijing University of Posts and Telecommunications (BUPT) in 2010. Now, he is an associate professor in Institute of Network Technology, BUPT. His research interests include cryptography, information security.

Wenmin Li received B.S. and M.S. degrees from Shaanxi Normal University in 2004 and 2007, respectively, and Ph.D. degree in cryptography from Beijing University of Posts and Telecommunications (BUPT) in 2012. Now, she is lecturer in Institute of Network Technology, BUPT. Her research interests include cryptography and information security.

Qiaoyan Wen received B.S. and M.S. degrees from Shaanxi Normal University in 1981 and 1984, respectively, and Ph.D. degree in cryptography from Xidian University in 1997. Now, she is a professor in Institute of Network Technology, BUPT. Her research interests include coding theory, cryptography, information security and applied mathematics.



Highlights

- We present an efficient homomorphic verifiable tag to improve computation efficiency.
- We propose a batch update algorithm to reduce the price of dynamic update.
- We describe an ODPDP scheme which is provably secure and highly efficient.
- The scheme migrates frequent auditing work from clients to an external auditor.
- The scheme provides a log audit mechanism with lower computation effort for clients.