On Verifying Dynamic Multiple Data Copies over Cloud Servers

Ayad F. Barsoum and M. Anwar Hasan

Department of Electrical and Computer Engineering
University of Waterloo, Ontario, Canada.
afekry@engmail.uwaterloo.ca, ahasan@uwaterloo.ca

August 15, 2011

Abstract

Currently, many individuals and organizations outsource their data to remote cloud service providers (CSPs) seeking to reduce the maintenance cost and the burden of large local data storage. The CSP offers paid storage space on its infrastructure to store customers' data. Replicating data on multiple servers across multiple data centers achieves a higher level of scalability, availability, and durability. The more copies the CSP is asked to store, the more fees the customers are charged. Therefore, customers need to be strongly convinced that the CSP is storing all data copies that are agreed upon in the service contract, and the data-update requests issued by the customers have been correctly executed on all remotely stored copies.

In this paper we propose two dynamic multi-copy provable data possession schemes that achieve two main goals: i) they prevent the CSP from cheating and using less storage by maintaining fewer copies, and ii) they support dynamic behavior of data copies over cloud servers via operations such as block modification, insertion, deletion, and append. We prove the security of the proposed schemes against colluding servers. Through theoretical analysis and experimental results, we demonstrate the performance of these schemes. Additionally, we discuss how to identify corrupted copies by slightly modifying the proposed schemes.

Index Terms

Cloud computing, outsourcing data storage, dynamic data integrity, cryptographic protocols

1 Introduction

Cloud computing (CC) is a computational model over a shared-virtualized pool of computing resources, e.g., storage, processing power, memory, applications, services, and network bandwidth. These days, CC has been receiving a great deal of attention from both academia and industry due to a number of important advantages it offers. One fundamental advantage of using CC is pay-as-you-go pricing model, where customers pay only according to their usage of the services [23]. Providing low management overhead and supporting immediate access to a wide range of applications are other principal benefits of adopting the CC technology. In addition, maintenance cost is reduced as a third party is responsible for everything from running the cloud to storing the data. Flexibility to scale up and down information technology (IT) capacity over time to business needs is also a key advantage of using the CC paradigm. Moreover, CC offers more mobility where customers can access information wherever they are, rather than having to remain at their desks.

Nowadays, many individuals and organizations outsource their data to remote cloud service providers (CSPs) relieving the load of constant server updates and other computing issues [35]. Such outsourcing of data storage enables customers to store more data on the CSP than on private computer systems. Also, it permits many authorized users to access the remotely stored data from different geographic locations making it more convenient for them. A relatively recent survey indicates that IT outsourcing has grown by a staggering 79% as organizations seek to focus more on their core competencies and reduce costs [36].

The data owners lose the control over their sensitive data once the latter is outsourced to a remote CSP which may not be trustworthy. This lack of control raises new formidable and challenging tasks related to data confidentiality and integrity protection in cloud computing. Customers require that their data remain secure over the CSP. Also, they need to have a strong evidence that the cloud servers still

possess the data and it is not being tampered with or partially deleted over time, especially because the internal operation details of the CSP may not be known to cloud customers.

Encrypting sensitive data before outsourcing to remote servers can handle the confidentiality issue. However, the integrity of customers' data in the cloud may be at risk due to the following reasons. First, the CSP — whose goal is likely to make a profit and maintain a reputation — has an incentive to hide data loss or reclaim storage by discarding data that has not been or is rarely accessed. Second, a greedy CSP might delete some of the data or might not store all data in fast storage required by the service contract with certain customers, i.e., place it on CDs or other offline media and thus making the access of data slower. Third, the cloud infrastructures are subject to a wide range of internal and external security threats. Examples of security breaches of cloud services appear from time to time [16], [21]. In short, although outsourcing data into the cloud is attractive from the view point of cost and complexity of long-term large-scale data storage, it does not offer any guarantee on data completeness and correctness. This problem, if not properly addressed, may hinder the successful deployment of cloud architecture.

Efficient verification techniques are of significant importance for cloud customers to validate data integrity over the CSP. Hash operations, signature schemes, and other traditional cryptographic primitives for data integrity and availability are not applicable on the outsourced data without having a local copy of the data. Downloading all outsourced data in order to validate its integrity is impractical for the excessive I/O cost and the high communication overhead across the network. Therefore, efficient techniques are needed to verify the integrity of outsourced data with minimum computation, communication, and storage overheads. Consequently, many researchers have focused on the problem of provable data possession (PDP) and proposed different schemes to audit the data stored on remote servers.

PDP is a technique for validating data integrity over remote servers. In a typical PDP model, the data owner generates some metadata/information for a data file to be used later for verification purposes through a challenge response protocol with the remote/cloud server. The owner sends the file to be stored on a remote server which may be untrusted, and deletes the local copy of the file. As a proof that the server is still possessing the data file in its original form, it correctly computes a response to a challenge vector sent from a verifier — who can be the original data owner or other trusted entity that shares some information with the owner. Researchers have proposed different variations of PDP schemes under different cryptographic assumptions; for example, see [2], [11], [15], [17], [26], [28], [30], [31], [37].

One of the core design principles of outsourcing data is to provide dynamic scalability of data for various applications. This means that the remotely stored data can be not only accessed by the authorized users, but also updated and scaled by the data owner. PDP schemes presented in [2], [11], [15], [17], [26], [28], [30], [31], [37] focus on *static* or warehoused data and do not consider the case of dynamic data that are usually more prevailing in practical applications.

Dynamic provable data possession (DPDP) constructions [3], [13], [19], [33], [34] reported in the literature focus on the provable possession of a *single* copy of a dynamic data file. Although PDP schemes have been presented for *multiple* copies of *static* data, e.g., [6], [10], [18], to the best of our knowledge, no PDP scheme exists for *multiple* copies of *dynamic* data.

■ Main contributions: Our contributions can be summarized as follows:

1) We propose two dynamic multi-copy provable data possession schemes. Through these schemes, the data owner can have an adequate guarantee that the CSP stores all copies that are agreed upon in the service contract. Moreover, the proposed schemes support outsourcing of dynamic data, i.e., they support operations such as block modification, insertion, deletion, and append. The authorized users — users who have the right to access the owner's file — can seamlessly access the copies received from the cloud servers.
2) We prove the security (correctness and soundness) of our schemes against colluding servers. Cloud servers can provide valid responses to verifier's challenges only if they actually have all data copies in an uncorrupted and updated state.
3) We present theoretical analysis and experimental results to justify the performance of the proposed schemes.

The remainder of the paper is organized as follows. Section 2 contains related work. Our system model

and proposed schemes are elaborated in Section 3. The performance analysis and experimental results are presented in Section 4. How to identify the corrupted copies is discussed in Section 5. Concluding remarks are given in Section 6.

2 Related Work

The key purpose of the PDP schemes is to validate the integrity of data outsourced to remote servers, where a verifier is allowed to efficiently, periodically, and securely validate that a remote server — which supposedly stores the owner's potentially very large amount of data — is actually storing the data in its original form. A simple solution to tackle the problem of data integrity over remote servers is by fragmenting the data file F into blocks $\{b_1, b_2, \ldots, b_m\}$, computing a message authentication code (MAC) σ_j for each block b_j: $\sigma_j = MAC_{sk}(j||b_j)_{1\leq j\leq m}$, sending both the data file F and the MACs $\{\sigma_j\}_{1\leq j\leq m}$ to the remote/cloud server, deleting the local copy of the file, and storing only the secret key sk. During the verification process, the verifier requests for a set of randomly selected blocks and their corresponding MACs, re-computes the MAC of each retrieved block using sk, and compares the re-computed MACs with the received values from the remote server [37]. This approach suffers from a severe drawback; the communication complexity is linear with the queried data size which is impractical especially when the available bandwidth is limited.

For efficient validation of outsourced data integrity, a number of PDP protocols have been proposed in the recent past [2], [11], [15], [17], [26], [28], [30], [31], [37]. But these schemes support only *static* data that are subject to infrequent change. Reference [6] conducted a literature survey and a comparative analysis of various static PDP schemes.

2.1 Hash-Based DPDP Schemes

Ateniese et al. [3] proposed a dynamic version of the PDP scheme based on cryptographic hash function and symmetric key encryption. Their scheme is efficient but allows only a fixed number of challenges due to the fact that through the scheme setup they come up with all future challenges and store pre-computed responses as tokens. These tokens can be stored either at the verifier side in a plain form or at the server side in an encrypted form. Block insertion in [3] cannot explicitly be supported (append operation is supported). Fig. 1 summarizes the scheme presented in [3].

2.2 DPDP Schemes Based on Authenticated Data Structures

2.2.1 Rank-Based Authenticated Skip Lists

A skip list is a hierarchical structure of linked lists [27], and is used to store a *sorted* set of items. Each node v in a normal skip list stores two pointers (right and down) denoted by $\mathsf{rgt}(v)$ and $\mathsf{dwn}(v)$ to be used during the searching procedure for a specific target in the leaf nodes (nodes at the base/bottom level). On the other hand, each node v in an *authenticated* skip list stores $\mathsf{rgt}(v)$, $\mathsf{dwn}(v)$, and a label $\mathsf{f}(v)$ computed by recursively applying a hash function to $\mathsf{f}(\mathsf{rgt}(v))$ and $\mathsf{f}(\mathsf{dwn}(v))$. The authenticated skip list can provide a proof to indicate whether a specific element belongs to the set represented by the list or not. In addition to $\mathsf{rgt}(v)$, $\mathsf{dwn}(v)$ and $\mathsf{f}(v)$, each node v in a *rank-based* authenticated skip list stores the number of nodes at the bottom level that can be reached from v. This number is called the rank of v: $\mathsf{r}(v)$.

Fig. 2 [13] shows an example of rank-based skip list, where the number inside the node represents its rank. The top leftmost node (w_7) of the skip list is considered to be the *start* node. To access any node at the bottom level, searching should begin from the start node.

2.2.2 DPDP Schemes of Erway et al.

Erway et al. [13] constructed a DPDP scheme based on the PDP model in [2] to support provable updates of stored data files using rank-based authenticated skip lists. Their protocol supports block insertion by eliminating the index information in the tag computation of [2]. The purpose of using the rank-based

Setup

- Data file F is a set of blocks $\{b_1, b_2, \ldots, b_m\}$
- g is a pseudo-random permutation, f is a pseudo-random function, and h is a cryptographic hash function. f is used to generate keys for g and to generate random numbers.
- E_K and E_K^{-1} are encryption and decryption algorithms under a key K
- Two master keys W and Z
- Data owner generates t random challenges and their corresponding responses/tokens $\{\nu_i\}_{1 \le i \le t}$ as follows.

  for i = 1 to t ***do***

  1. Generate $k_i = f_W(i)$ and $c_i = f_Z(i)$
  2. $\nu_i = h(c_i, 1, b_{g_{k_i}(1)}) \oplus \cdots \oplus h(c_i, r, b_{g_{k_i}(r)})$ /* r is # of blocks per token */
  3. $\nu_i' = E_k(ctr, i, \nu_i)$ /* ctr is an integer counter */

- Owner sends the file $F = \{b_j\}_{1 \le j \le m}$ and $\{\nu_i'\}_{1 \le i \le t}$ to the remote server.

Challenge Response

Data owner — Remote Server

Begin challenge i

1. Generates $k_i = f_W(i)$ and $c_i = f_Z(i)$

$\xrightarrow{k_i, c_i}$

2. $z = h(c_i, 1, b_{g_{k_i}(1)}) \oplus \cdots \oplus h(c_i, r, b_{g_{k_i}(r)})$

$\xleftarrow{z, \nu_i'}$

3. Computes $\nu = E_K^{-1}(\nu_i')$
4. Checks $\nu \stackrel{?}{=} (ctr, i, z)$

End challenge i

DYNAMIC OPERATIONS

Modify

/* *Assume that block b_j is to be updated to b_j'* */

Data owner — Remote Server

$\xleftarrow{1. \{\nu_i'\}_{1 \le i \le t}}$

2. $ctr = ctr + 1$
3. ***for*** i = 1 to t ***do***

   3.1 $z_i' = E_K^{-1}(\nu_i')$

   /* *if decryption fails, exit** */

   /* *if z_i' is not prefixed by (ctr-1) and i, exit* */

   3.2 extracts ν_i from z_i'

   3.3 computes $k_i = f_W(i)$ and $c_i = f_Z(i)$

   3.4 ***for*** l = 1 to r ***do***

   if$(g_{k_i}(l) == j)$ ***then***

   $\nu_i = \nu_i \oplus h(c_i, l, b_j) \oplus h(c_i, l, b_j')$

   /**update all tokens even if they do not include the block to be updated* */

   3.5 $\nu_i' = E_k(ctr, i, \nu_i)$

$\xrightarrow{j,\ b_j',\ \{\nu_i'\}_{1 \le i \le t}}$

$\Longrightarrow$ ***continue***

Delete
/ Assume that block b_j is to be deleted */*
The logic of **Delete** is similar to **Modify** operation but replaces the block to be modified with a special block "DBlock". So, the inner ***for*** loop (step 3.4) will be:
3.4 ***for*** l = 1 to r ***do***
if$(g_{k_i}(l) == j)$ ***then***
$\nu_i = \nu_i \oplus h(c_i, l, b_j) \oplus h(c_i, l, \mathsf{DBlock})$
Insert
Physical insert is not supported. Append operation is allowed by viewing the data file as a two dimensional structure (matrix), and appending the new block in a round robin fashion.

Fig. 1: The DPDP protocol by Ateniese et al. [3].

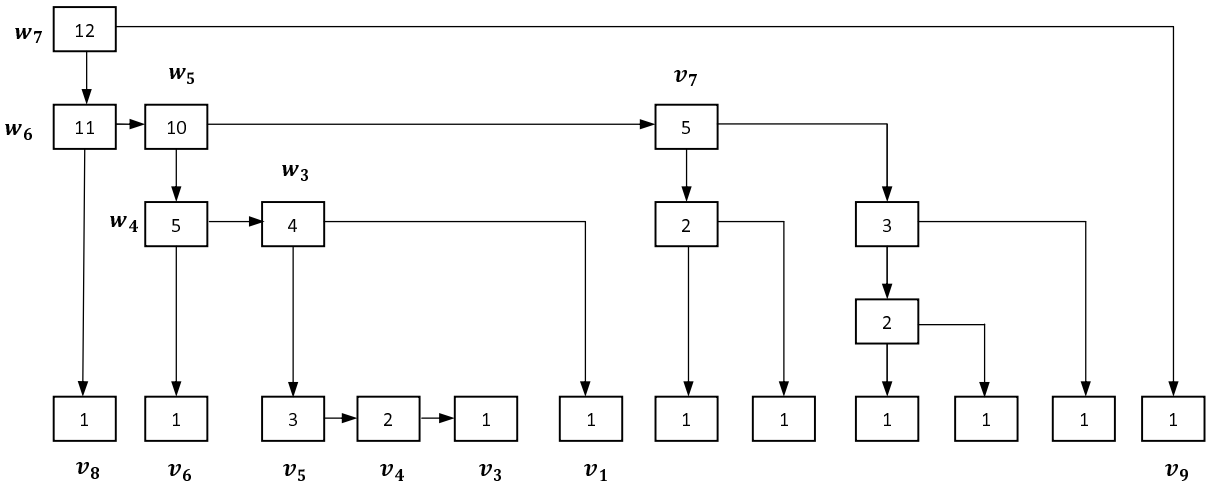


Fig. 2: Example of rank-based skip list [13].

authenticated skip list in [13] is to authenticate the tag information of the blocks to be updated or challenged.

In the DPDP scheme of [13], the File F is fragmented into m blocks $\{b_1, b_2, \ldots, b_m\}$. A representation/tag $\mathcal{T}(b_j)$ of block b_j is computed as $\mathcal{T}(b_j) = g^{b_j} \bmod N$ (N is the RSA modulus and g is an element of high order in $\mathbb{Z}_N^*$). The block representation $\mathcal{T}(b_j)$ is stored at the j^{th} bottom-level node of the authenticated skip list and the block itself is stored elsewhere by the server. The tags protect the integrity of file blocks, while the authenticated list ensures the security and integrity of tags.

During the challenge phase, the client requests the server to prove the integrity of randomly selected c blocks $\{b_{j_i}\}_{1 \le j_1, \ldots, j_c \le m}$. The server sends the tags $\{\mathcal{T}(b_{j_i})\}_{1 \le j_1, \ldots, j_c \le m}$ along with their search/verification paths. Also, the server sends a combined block $M = \sum_{i=1}^{c} a_i.b_{j_i}$, where $\{a_i\}_{1 \le i \le c}$ are random values sent by the client as part of the challenge. The owner verifies the search/verification paths of the block tags using the metadata $\mathcal{M}_c$ which is the label of the start node. Besides, the owner computes $T = \prod_{i=1}^{c} \mathcal{T}(b_{j_i})^{a_i} \bmod N$. The owner accepts only if the search paths are verified and $T = g^M \bmod N$.

The authenticated skip list is used to modify, insert, and delete the block tags achieving the dynamic behavior of the data file. Nodes of skip list along the search/verification path — path from the start node to the node associated with the block to be updated — are only affected by the dynamic operations of file blocks.

The scheme presented in [13] can be summarized by the following procedures:

- *Key-generation*. This procedure is run by the data owner (the client) and outputs a secret key sk and public key pk. The secret key is kept by the client and the public key is sent to the remote server.
- *Update-preparation*. This procedure is run by the client to prepare a part of the file for storage on a remote server which may not be trustworthy. The input parameters to this procedure are sk, pk, a

part of the file F, updates to be performed (e.g., full re-write, modify block, delete block, or insert block), and the previous metadata $\mathcal{M}_c$. The output is an encoded version of that part of the file $\mathsf{e}(F)$, encoded information $\mathsf{e}(\mathsf{info})$ about the update, and the new metadata $\mathsf{e}(\mathcal{M})$. The output of this procedure is sent to the remote server.

- *Update-execution*. Upon receiving an update request from the data owner, the server runs this procedure in response to the owner's request. The input parameters to this procedure are pk, the previous version of the file denoted as F_{i-1}, the metadata $\mathcal{M}_{i-1}$, and the values produced by the client during the *Update-preparation* algorithm. The outputs of this procedure are the new version of the file denoted as F_i , metadata $\mathcal{M}_i$, and metadata $\mathcal{M}'_c$ to be sent to the owner along with its proof $P_{\mathcal{M}'_c}$.
- *Update-verification*. This procedure is run by the client to verify the server's behavior during the updates. The input parameters to this procedure are all inputs of *Update-preparation* algorithm, the metadata $\mathcal{M}'_c$, and the proof $P_{\mathcal{M}'_c}$ ($\mathcal{M}'_c$ and $P_{\mathcal{M}'_c}$ are sent from the server as outputs of the *Update-execution* algorithm). The output of the *Update-verification* algorithm is either acceptance or rejection signal.
- *Challenge*. This procedure is run by the client to challenge the server and verify the integrity of the remotely stored data file. It takes as input sk, pk, and the latest client metadata $\mathcal{M}_c$. The output is a challenge c that is sent from the client to the server.
- *Proof-computation*. Upon receiving the challenge c from the client, the server runs the *Proof-computation* algorithm in response to the owner's challenge. The input parameters to this procedure are pk, the latest version of the file, the metadata, and the challenge c. It outputs a proof P that is sent to the client.
- *Proof-verification*. This procedure is run by the client to validate the proof P received from the server. The input parameters to this procedure are sk, pk, the client metadata $\mathcal{M}_c$, the challenge c, and the proof P sent by the server. The output of this procedure is "accept" to indicate that the server still possesses the file intact or "reject" otherwise.

In their paper, Erway et al. [13] presented a variant of the DPDP scheme using RSA trees instead of rank-based authenticated lists. Wang et al. [34] used Merkle hash trees [25] (instead of skip lists) and homomorphic authenticators built from BLS signatures [7] to construct a DPDP scheme.

2.3 RSA-Based DPDP Schemes

Hao et al. [19] adapted the protocol proposed in [28] to support both the dynamic nature of data and public verifiability. Public verifiability allows anyone, who knows the owner's public key but is not necessarily the data owner, challenge the remote server and verify that the server is still possessing the owner's files. If a disagreement regarding data integrity happens between the owner and the CSP, a third party auditor (TPA) can determine whether the data integrity is maintained or not. This third party auditing process should bring in no new vulnerabilities towards the privacy of owner's data. The protocol proposed in [19] ensures that the data is kept private during the third party verification, where no private information contained in the data is leaked. Fig. 3 summarizes the protocol presented in [19].

2.4 Cooperative PDP Schemes

Zhu et al. [38] addressed the construction of cooperative PDP scheme on hybrid clouds to support scalability of service and data migration. A hybrid cloud is a CC deployment model in which an organization provides and handles some internal and external resources. For example, an organization can use a public cloud service like Amazon EC2 [1] to perform the general computation, while the data files are stored within the organization's local data center in a private cloud. In their work, Zhu et al. [38] consider the existence of multiple CSPs that cooperatively store customers' data

Remark 1. The previous schemes reported in [3], [13], [19], [34] focus on the data dynamic of a *single* copy. References [10], [6], and [18] consider the case of multiple data copies over remote servers, but they focus on *static* archived data. In this work, we study the integrity verification of *dynamic multiple* data copies stored on untrusted cloud servers.

Setup

- $N = pq$ is the RSA modulus (p and q are prime numbers)
- g is a generator of Q_{RN} (Q_{RN} is the set of quadratic residues modulo N)
- Public key $pk = (N, g)$ and secret key $sk = (p, q)$.
- f is a pesudo-random function
- File $F = \{b_1, b_2, \ldots, b_m\}$.
- Data owner generates a tag D_j for each block b_j, where $D_j = g^{b_j} \bmod N$
- The tags are stored on the owner side and the file is sent to the remote server.

Challenge Response

Verifier — Remote Server

1. Generates a random key r and $s \in_R \mathbb{Z}_N$
2. Computes $g_s = g^s \bmod N$

$\xrightarrow{r, g_s}$

3. Generates random coefficients $\{a_j = f_r(j)\}_{1 \le j \le m}$
4. Computes $R = (g_s)^{\sum_{j=1}^{m} a_j . b_j} \bmod N$

$\xleftarrow{R}$

5. Generates a set of random coefficients $\{a_j = f_r(j)\}_{1 \le j \le m}$
6. Computes $P = \prod_{j=1}^{m} (D_j^{a_j} \bmod N) \bmod N$
7. Computes $R' = P^s \bmod N$
8. Checks $R' \stackrel{?}{=} R$

DYNAMIC OPERATIONS

Modify

/ Assume that block b_j is to be updated to b'_j */*

- Server updates b_j to b'_j
- Owner computes a new block tag $D'_j = g^{b'_j} \bmod N$.
  So, the new block tags are $\{D_1, D_2, \ldots, D'_j, \ldots, D_m\}$

Insert

/ Assume a new block $\hat{b}$ is to be inserted after position j or appended at the end */*

- The server updates its file to be
  $\{b_1, b_2, \ldots, b_j, \hat{b}, \ldots, b_{m+1}\}$ (insert & $b_{j+1} = \hat{b}$) or $\{b_1, b_2, \ldots, b_m, \hat{b}\}$ (append).
- The owner computes a new block tag $\hat{D} = g^{\hat{b}} \bmod N$, and changes the block tags to $\{D_1, D_2, \ldots, D_j, \hat{D}, \ldots, D_{m+1}\}$ (insert & $D_{j+1} = \hat{D}$) or $\{D_1, D_2, \ldots, D_m, \hat{D}\}$ (append)

Delete

/ Assume a block at position j is to be deleted*/*

- Server deletes the block b_j
- Owner deletes the corresponding tag D_j

Fig. 3: The DPDP protocol by Hao et al. [19].

2.5 Proof of Retrievability

Proof of retrievability (POR) is a complementary approach to PDP that allows a verifier to check the possession of data via a challenge response protocol. POR is stronger than PDP in the sense that the verifier can reconstruct the entire data file from the responses that are reliably transmitted from the server. This is due to encoding of the data file, for example using *erasure codes*, before outsourcing to remote servers. Various POR schemes can be found in the literature, for example [8], [9], [12], [20], [29], which focus on *static* data.

In the context of this work, we are dealing with dynamic data in which the owner can update and

scale the files stored by the CSP. Thus, if the data file is encoded before outsourcing, modifying a portion of the file requires re-encoding the data file which is not acceptable in practical applications. In addition, we are considering economically-motivated CSPs that may attempt to use less storage than required by the contract through deletion of a few copies of the file. The CSPs have almost no financial benefit by deleting only a small portion of a copy of the file. As a result, in our work we do not encode the data file before outsourcing. Besides, a server's copy can be reconstructed even from a complete damage using duplicated copies on other servers.

More importantly, unlike erasure codes, duplicating data file across multiple servers achieves scalability which is a fundamental customer requirement in CC systems. A file that is duplicated and stored strategically on multiple servers — located at various geographic locations — can help reduce access time and communication cost for users.

3 Proposed Dynamic Multi-Copy Provable Data Possession Schemes

3.1 Cloud Data Storage Model

The cloud computing system model considered in this work consists of three main components as illustrated in Fig. 4: (i) a data owner that can be an individual or an organization originally possessing sensitive data to be stored in the cloud; (ii) a CSP who manages cloud servers (CSs) and provides paid storage space on its infrastructure to store the owner's files; and (iii) authorized users — a set of owner's clients who have the right to access the remote data and share some keying material with the data owner. An authorized user sends a data request to the CSP and receives a data copy in an encrypted form that can be decrypted using a secret key shared with the owner.

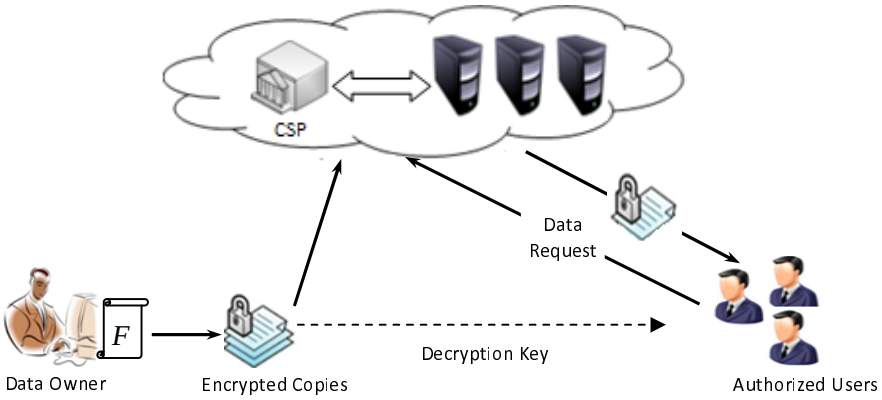


Fig. 4: Cloud computing data storage system model.

We assume that the interaction between the owner and the authorized users to authenticate their identities and share the secret key has already been completed, and it is not considered in this work. Throughout this paper, the terms cloud server and cloud service provider are used interchangeably.

3.2 Context and Rationale

Homomorphic linear authenticators (HLAs) are basic building blocks in the proposed schemes. The HLA is a fingerprint/tag for each file block b_j that enables a verifier to validate the data possession on a cloud server by sending a challenge vector $C = \{c_1, c_2, \ldots, c_r\}$. As a response, the server can homomorphically construct a tag authenticating the value $\sum_{j=1}^{r} c_j \cdot b_j$. The proposed schemes in this work utilize the BLS HLAs [29].

Generating unique differentiable copies of the data file is the main core to design a multi-copy provable data possession scheme. Identical data copies enable the CSP to simply deceive the owner by storing only one copy and pretending that it stores multiple copies. Using a simple yet *efficient* way, the proposed schemes generate distinct copies utilizing the *diffusion* property of any secure encryption scheme. According to the diffusion property, the output bits of the ciphertext should depend on the input bits of

the plaintext in a very complex way, i.e., there will be an unpredictable complete change in the ciphertext, if there is a single bit change in the plaintext [32]. The interaction between the authorized users and the CSP is considered through this methodology of generating distinct copies, where the former can decrypt and access a file copy received from the CSP. In the proposed schemes, the authorized users need only to keep a single secret key — shared with the data owner — to decrypt the file copy; it is not necessarily to recognize the index of the received copy.

In this work, the data owner has a file F and the CSP offers to store n copies $\{\widetilde{F}_1, \widetilde{F}_2, \ldots, \widetilde{F}_n\}$ of the owner's file in exchange of pre-specified fees. We provide two schemes: tree-based and map-based dynamic multi-copy provable data possession (TB-DMCPDP and MB-DMCPDP, respectively). The TB-DMCPDP scheme can be viewed as a *fine-tuned* extension to the provable possession of dynamic single-copy models that are based on authenticated data structures, e.g., [13] and [34].

3.3 Notations

- F is a data file to be outsourced, and is composed of a sequence of m blocks, i.e., $F = \{b_1, b_2, \ldots, b_m\}$.
- $\pi_{key}(\cdot)$ is a pseudo-random permutation (PRP): $key \times \{0,1\}^{\log(m)} \rightarrow \{0,1\}^{\log(m)}$. [1]
- $\psi_{key}(\cdot)$ is a pseudo-random function (PRF): $key \times \{0,1\}^* \rightarrow \mathbb{Z}_p$.
- **Bilinear Map/Pairing.** The bilinear map is one of the essential building blocks of the proposed schemes. Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p. Let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. A bilinear pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the following properties [24]:
  1) *Bilinear*: $\hat{e}(u_1 u_2, v_1) = \hat{e}(u_1, v_1) \cdot \hat{e}(u_2, v_1)$, $\hat{e}(u_1, v_1 v_2) = \hat{e}(u_1, v_1) \cdot \hat{e}(u_1, v_2)$ $\forall$ $u_1, u_2 \in \mathbb{G}_1$ and $v_1, v_2 \in \mathbb{G}_2$
  2) *Non-degenerate*: $\hat{e}(g_1, g_2) \neq 1$
  3) *Computable*: there exists an efficient algorithm for computing $\hat{e}$
  4) $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$ $\forall$ $u \in \mathbb{G}_1, v \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$
- h is a cryptographic hash function, e.g., SHA-2.
- $\mathcal{H}(\cdot)$ is a map-to-point hash function : $\{0,1\}^* \rightarrow \mathbb{G}_1$.
- E_K is an encryption algorithm with strong *diffusion* property, e.g., AES.

3.4 Underlying Algorithms

The proposed schemes consist of seven polynomial time algorithms: KeyGen, CopyGen, TagGen, PrepareUpdate, ExecUpdate, Prove, and Verify.

- $(pk, sk) \leftarrow$ KeyGen(). This algorithm is run by the data owner to generate public key pk and private key sk.
- $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}_{1 \leq i \leq n} \leftarrow$ CopyGen$(CN_i, F)_{1 \leq i \leq n}$. This algorithm is run by the data owner. It takes as input a copy number CN_i and a file F, and generates unique differentiable copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}_{1 \leq i \leq n}$. A file copy $\widetilde{F}_i$ is an ordered collection of blocks $\{\tilde{b}_{ij}\}_{1 \leq j \leq m}$, where each block $\tilde{b}_{ij}$ is fragmented into s sub-blocks (sectors): $\tilde{b}_{ij} = \{\tilde{b}_{ij1}, \tilde{b}_{ij2}, \ldots, \tilde{b}_{ijs}\}$. Thus, $\widetilde{F}_i = \{\tilde{b}_{ijk}\}_{\substack{1 \leq j \leq m \\ 1 \leq k \leq s}}$, where each sector $\tilde{b}_{ijk} \in \mathbb{Z}_p$ for some large prime p.
- $\Phi \leftarrow$ TagGen$(sk, \widetilde{\mathbb{F}})$. This algorithm is run by the data owner. It takes as input the private key sk and the unique file copies $\widetilde{\mathbb{F}}$, and outputs the tags/authenticators set Φ, which is an ordered collection of tags for the data blocks. Tags are generated per block not per sector to minimize the storage overhead.
- *Updateinfo* $\leftarrow$ PrepareUpdate(). This algorithm is run by the data owner to update the outsourced file copies stored by the remote CSP. The output of this algorithm is *Updateinfo*: information about the update request. This information may contain a modified version of a previously stored block, a new

1. The number of file blocks (m) will be changed due to dynamic operations on the file. We use HMAC-SHA-1 with 160-bit output, to allow up to 2^{160} blocks in the file.

block to be inserted, or a delete command to delete a specific block from the file copies. *Updateinfo* is sent from the data owner to the CSP in order to perform the requested update.

– ($\widetilde{\mathbb{F}}'$, Φ', *auth*) $\leftarrow$ ExecUpdate($\widetilde{\mathbb{F}}$, Φ, *Updateinfo*). This algorithm is run by the CSP, where the input parameters are the file copies $\widetilde{\mathbb{F}}$, the tag set Φ, and *Updateinfo* (sent from the owner). It outputs an updated version of the file copies $\widetilde{\mathbb{F}}'$ along with an updated signature set Φ' and some authentication information *auth* [2].

– $\mathbb{P} \leftarrow$ Prove($\widetilde{\mathbb{F}}, \Phi$). This algorithm is run by the CSP. It takes as input the file copies $\widetilde{\mathbb{F}}$ and the tag set Φ, and returns a proof $\mathbb{P}$ which guarantees that the CSP is actually storing n copies and all these copies are intact, updated, and consistent.

– $\{1, 0\} \leftarrow$ Verify($pk, \mathbb{P}$). Our schemes support public verifiability, so this algorithm can be run by any verifier (data owner or any other trusted auditor). It takes as input the public key pk and the proof $\mathbb{P}$ returned from the CSP, and outputs 1 if the integrity of all file copies is correctly verified or 0 otherwise.

3.5 The Proposed TB-DMCPDP Scheme

The TB-DMCPDP scheme proposed in this work is based on using Merkle hash trees (MHTs) [25], but it can also be constructed using authenticated skip lists [13] or other authenticated data structures.

3.5.1 Merkle Hash Tree

A MHT [25] is a binary tree structure used to efficiently verify the integrity of the data. The MHT is a tree of hashes where the leaves of the tree are the hashes of the data blocks. Fig. 5 shows an example of a MHT used to verify the integrity of a file F consisting of 8 blocks. $h_j = h(b_j)$ $(1 \leq j \leq 8)$. $h_A = h(h_1||h_2)$,

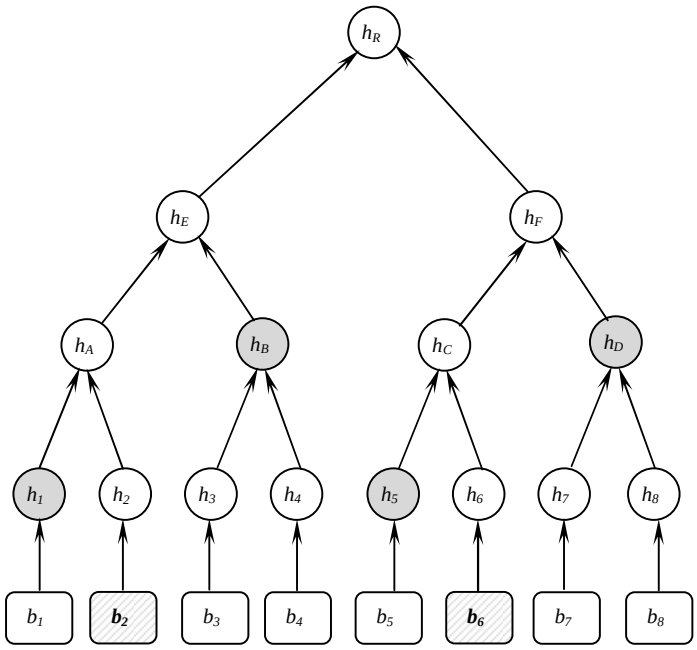


Fig. 5: Merkle Hash Tree.

$h_B = h(h_3||h_4)$, and so on. Finally, $h_R = h(h_E||h_F)$ is the hash of the root node that is used to authenticate the integrity of all data blocks. The data blocks $\{b_1, b_2, \ldots, b_8\}$ are stored on a remote server, and only the authentic value h_R is stored locally on the verifier side. For example, if the verifier requests to check the integrity of the blocks b_2 and b_6, the server will send these two blocks along with the authentication paths $\mathbb{A}_2 = \{h_1, h_B\}$ and $\mathbb{A}_6 = \{h_5, h_D\}$ that are used to reconstruct the root of the MHT. $\mathbb{A}_j$ — the authentication path of b_j — is a set of node siblings (grey-shaded circles) on the path from h_j to the root

2. The *auth* parameter in the MB-DMCPDP scheme is *null* and not used

of the MHT. The verifier uses the received blocks and the authentication paths to recompute the root in the following manner. The verifier constructs $h_2 = h(b_2)$, $h_6 = h(b_6)$, $h_A = h(h_1||h_2)$, $h_C = h(h_5||h_6)$, $h_E = h(h_A||h_B)$, $h_F = h(h_C||h_D)$, and $h_R = h(h_E||h_F)$. After computing h_R, it is compared with the authentic value stored locally on the verifier side.

The MHT is commonly used to authenticate the *values* of the data blocks. In the dynamic behavior of outsourced data, we need to authenticate both the *values* and the *positions* of the data blocks, i.e., we need an assurance that a specific value is stored at a specific leaf node. For example, if a data owner requires to insert a new block after position j, the verifier needs to make sure that the server has inserted the new block in the requested position. To validate the positions of the blocks, the leaf nodes of the MHT are treated in a specific sequence, e.g., left-to-right sequence [22]. So, the hash of any internal node = h(left child || right child), e.g., $h_A = h(h_1||h_2) \neq h(h_2||h_1)$. Besides, the authentication path $\mathbb{A}_j$ is viewed as an *ordered* set, and thus any leaf node is uniquely specified by following the used sequence of constructing the root of the MHT.

3.5.2 Directory MHT for File Copies

The proposed schemes in this paper are aiming at supporting multiple copies of dynamic data. To this end, in the TB-DMCPDP scheme a MHT is constructed for each file copy, and then the roots of the individual trees are used to build a hash tree which we call a directory MHT. The key idea is to make the root node of each copy's MHT as a leaf node in a directory MHT used to authenticate the integrity of all file copies in a hierarchical manner. The directory tree is depicted in Fig. 6. The verifier can keep only one metadata $\mathcal{M} = h(ID_F||h_{DR})$, where ID_F is a unique file identifier for each owner's file, and h_{DR} is the authenticated directory root value that can be used to periodically check the integrity of all file copies. Embedding the ID_F into the metadata $\mathcal{M}$ prevents the CSP from cheating using blocks from different files.

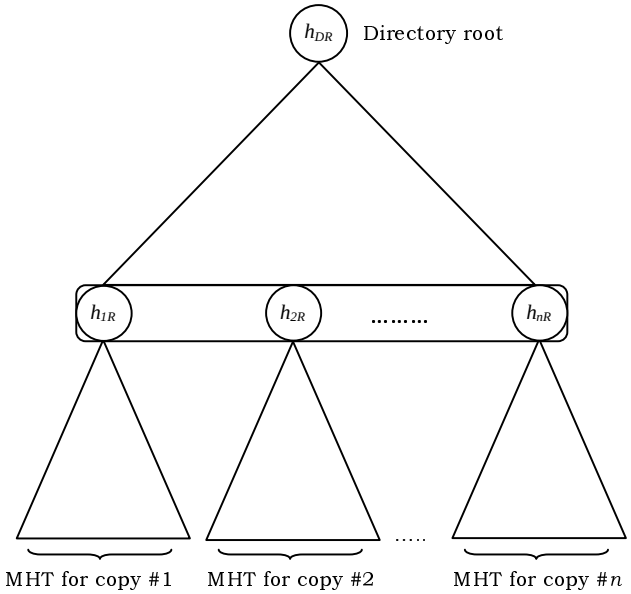


Fig. 6: Directory tree for the file copies.

3.5.3 TB-DMCPDP Procedural Steps

The procedures of our TB-DMCPDP scheme execution are as follows:

- ***Key Generation***. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a bilinear map, $(u_1, u_2, \ldots, u_s) \in_R \mathbb{G}_1$, and g is a generator of $\mathbb{G}_2$. The data owner runs the KeyGen algorithm to generate a private key $x \in \mathbb{Z}_p$ and a public key $y = g^x \in \mathbb{G}_2$.
- ***Generation of Distinct Copies***. The data owner runs the CopyGen algorithm to generate n differentiable copies. For a file $F = \{b_1, b_2, \ldots, b_m\}$, The CopyGen algorithm creates a unique copy $\widetilde{F}_i = \{\tilde{b}_{ij}\}$

$(1 \leq i \leq n, 1 \leq j \leq m)$ by concatenating a copy number i with the block b_j, then encrypting using an encryption scheme E_K, i.e., $\tilde{b}_{ij} = E_K(i||b_j)$. The encrypted block $\tilde{b}_{ij}$ is fragmented into s sectors, i.e., $\widetilde{F}_i = \{\tilde{b}_{ijk}\}_{\substack{1\leq j\leq m \\ 1\leq k\leq s}}$, where each sector $\tilde{b}_{ijk} \in \mathbb{Z}_p$ for some large prime p. The authorized users need only to keep a single secret key K. Later, when an authorized user receives a file copy from the CSP, he decrypts the copy blocks, removes the copy index from the blocks header, and then recombines the decrypted blocks to reconstruct the plain form of the received file copy.

- ***Generation of Tags and Trees***. Given the distinct file copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}$, where $\widetilde{F}_i = \{\tilde{b}_{ijk}\}$, the data owner runs the TagGen algorithm to create a tag σ_{ij} for each block $\tilde{b}_{ij}$ as $\sigma_{ij} = (\mathcal{H}(\tilde{b}_{ij}) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x \in \mathbb{G}_1$ $(i : 1 \to n,\ j : 1 \to m,\ k : 1 \to s)$. The data owner then generates an aggregated tag σ_j for the blocks at the same indices in each copy $\widetilde{F}_i$ as $\sigma_j = \prod_{i=1}^{n} \sigma_{ij} \in \mathbb{G}_1$. We denote the set of aggregated tags as $\Phi = \{\sigma_j\}_{1\leq j\leq m}$.
  The data owner generates a MHT for each file copy $\widetilde{F}_i$. The leaf nodes of each tree are the ordered set $\{h(\mathcal{H}(\tilde{b}_{ij}))\}$, i.e., the leaf nodes of the MHT are the cryptographic hashes of $\mathcal{H}(\tilde{b}_{ij})$, and the root of the tree is denoted as h_{Ri}. Using the roots $\{h_{Ri}\}_{1\leq i\leq n}$, the data owner generates a directory MHT in which the leaf nodes are $\{h_{Ri}\}_{1\leq i\leq n}$, and the directory root is denoted as h_{DR}. Note that the MHTs are constructed using a specific sequence, e.g., left-to-right sequence to authenticate both the value and the position of $\mathcal{H}(\tilde{b}_{ij})$. The owner computes a metadata $\mathcal{M} = h(ID_F||h_{DR})$, where $ID_F = Filename||n||u_1||\ldots||u_s$, i.e., the ID_F is a unique fingerprint for each file comprising the file name, the number of copies for this file, and the random values $\{u_k\}_{1\leq k\leq s}$. The data owner sends $\langle\widetilde{\mathbb{F}}, \Phi, ID_F, \{MHT_i\}_{1\leq i\leq n}\rangle$ to the CSP and deletes the copies, the tags, and the trees from its local storage. The metadata $\mathcal{M}$ is stored on the local storage of the owner (or any trusted verifier).
- ***Dynamic Operations on the Data Copies***. The dynamic operations in the proposed TB-DMCPDP scheme are performed on the block level via a request in the general form $\langle ID_F, \mathit{BlockOp}, j, \{b_i^*\}_{1\leq i\leq n}, \sigma_j^*\rangle$, where ID_F is the file identifier, and *BlockOp* corresponds to block modification (denoted by BM), block insertion (denoted by BI), or block delete (denoted by BD). The parameter j indicates the index of the block to be updated, $\{b_i^*\}_{1\leq i\leq n}$ are the new block values for all copies, and σ_j^* is the new aggregated tag for the new blocks.
  - **Modification**. Data modification is one of the most frequently used dynamic operations in the outsourced data. For a file $F = \{b_1, b_2, \ldots, b_m\}$, suppose the owner wants to modify the block b_j with the block b'_j for all file copies. The owner runs the PrepareUpdate algorithm to do the following:
    1) Creates n distinct blocks $\{\tilde{b}'_{ij}\}_{1\leq i\leq n}$, where $\tilde{b}'_{ij} = E_K(i||b'_j)$ is fragmented into s sectors $\{\tilde{b}'_{ij1}, \tilde{b}'_{ij2}, \ldots, \tilde{b}'_{ijs}\}$
    2) Creates a new tag σ'_{ij} for each block $\tilde{b}'_{ij}$ as $\sigma'_{ij} = (\mathcal{H}(\tilde{b}'_{ij}) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}'_{ijk}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\sigma'_j = \prod_{i=1}^{n} \sigma'_{ij} \in \mathbb{G}_1$
    3) Sends a modify request $\langle ID_F, \mathsf{BM}, j, \{\tilde{b}'_{ij}\}_{1\leq i\leq n}, \sigma'_j\rangle$ to the CSP

    Upon receiving the modify request from the data owner, the CSP runs the ExecUpdate algorithm to do the following:
    1) Replaces the block $\tilde{b}_{ij}$ with $\tilde{b}'_{ij} \forall i$, and constructs an updated version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1\leq i\leq n}$
    2) Replaces $h(\mathcal{H}(\tilde{b}_{ij}))$ with $h(\mathcal{H}(\tilde{b}'_{ij}))$ in the leaf nodes of each copy's MHT, and accordingly updates the MHTs
    3) Calculates the authentication paths $\langle\mathbb{A}_{ij}\rangle_{1\leq i\leq n}$ of the updated blocks at position j in all copies. $\mathbb{A}_{ij}$ is an *ordered* set of node siblings on the path from the leaf node $h(\mathcal{H}(\tilde{b}'_{ij}))$ to the root of the MHT of copy i
    4) Replaces σ_j with σ'_j in the aggregated tags set Φ, and outputs $\Phi' = \{\sigma_1, \sigma_2, \ldots, \sigma'_j, \ldots, \sigma_m\}$
    5) Sends $\langle\mathbb{A}_{ij}\rangle_{1\leq i\leq n}$ to the owner

    Upon receiving $\langle\mathbb{A}_{ij}\rangle_{1\leq i\leq n}$ from the CSP, the owner uses these authentication paths and $\{\tilde{b}'_{ij}\}_{1\leq i\leq n}$

to generate a new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$. The interaction between the data owner and the CSP during block modification is summarized in Fig. 7.

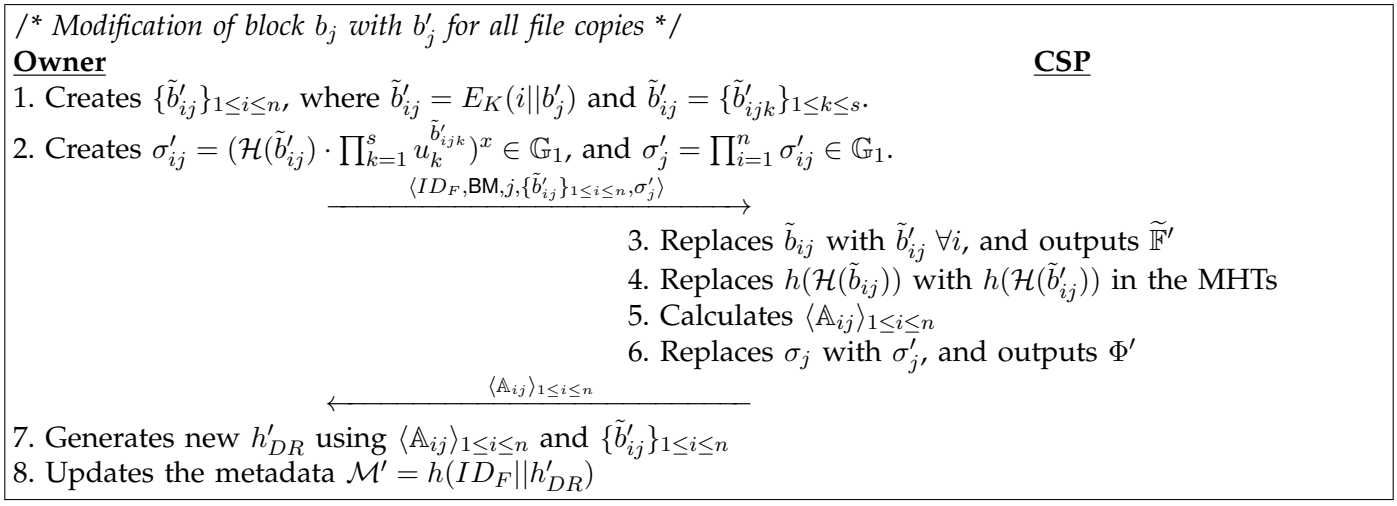


Fig. 7: The block modification protocol in the TB-DMCPDP scheme.

- **Insertion**. In the block insertion operation, the owner wants to insert a new block $\hat{b}$ *after* position j in a file $F = \{b_1, b_2, \ldots, b_m\}$, i.e., the newly constructed file $F' = \{b_1, b_2, \ldots, b_j, \hat{b}, \ldots, b_{m+1}\}$, i.e., $b_{j+1} = \hat{b}$. The block insertion operation changes the logical structure of the file, while block modification does not. In the proposed TB-DMCPDP scheme, the block index is not included in the block tag. Thus, the insertion operation can be performed without recomputing the tags of all blocks that have been shifted after inserting the new block. Embedding the block index in the tag results in unacceptable computation overhead, especially for large data files. MHTs are used to validate the positions of file blocks. To perform the insertion of a new block $\hat{b}$ after position j in all file copies, the owner runs the PrepareUpdate algorithm to do the following:
  1) Creates n distinct blocks $\{\hat{b}_i\}_{1\le i\le n}$, where $\hat{b}_i = E_K(i||\hat{b})$ is fragmented into s sectors $\{\hat{b}_{i1}, \hat{b}_{i2}, \ldots, \hat{b}_{is}\}$
  2) Creates a new tag $\hat{\sigma}_i$ for each block $\hat{b}_i$ as $\hat{\sigma}_i = (\mathcal{H}(\hat{b}_i) \cdot \prod_{k=1}^{s} u_k^{\hat{b}_{ik}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\hat{\sigma} = \prod_{i=1}^{n} \hat{\sigma}_i \in \mathbb{G}_1$
  3) Sends an insert request $\langle ID_F, \mathsf{BI}, j, \{\hat{b}_i\}_{1\le i\le n}, \hat{\sigma}\rangle$ to the CSP

  Upon receiving the insert request from the owner, the CSP runs the ExecUpdate algorithms to do the following:
  1) Inserts the block $\hat{b}_i$ after position j in the file copy $\widetilde{F}_i$ $\forall i$, and then adds a leaf node $h(\mathcal{H}(\hat{b}_i))$ after the leaf node $h(\mathcal{H}(\tilde{b}_{ij}))$ for each copy's MHT. This leads to constructing a new version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1\le i\le n}$, and a new version of the MHTs
  2) Calculates the authentication paths $\langle \widehat{\mathbb{A}}_i\rangle_{1\le i\le n}$ of the newly inserted blocks $\{\hat{b}_i\}_{1\le i\le n}$ in all copies. $\widehat{\mathbb{A}}_i$ is an ordered set of node siblings on the path from the leaf node $h(\mathcal{H}(\hat{b}_i))$ to the root of the MHT of copy i
  3) Inserts $\hat{\sigma}$ after position j in the set of the aggregated tags Φ, and outputs $\Phi' = \{\sigma_1, \sigma_2, \ldots, \sigma_j, \hat{\sigma}, \ldots, \sigma_{m+1}\}$, i.e., $\sigma_{j+1} = \hat{\sigma}$
  4) Sends $\langle \widehat{\mathbb{A}}_i\rangle_{1\le i\le n}$ to the owner

  Upon receiving $\langle \widehat{\mathbb{A}}_i\rangle_{1\le i\le n}$ from the CSP, the owner uses these authentication paths and $\{\hat{b}_i\}_{1\le i\le n}$ to generate a new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$. The interaction between the data owner and the CSP during block insertion is summarized in Fig. 8.

- **Append**. Block append operation means adding a new block at the end of the outsourced data. It can simply be implemented via insert operation after the last block of the data file.

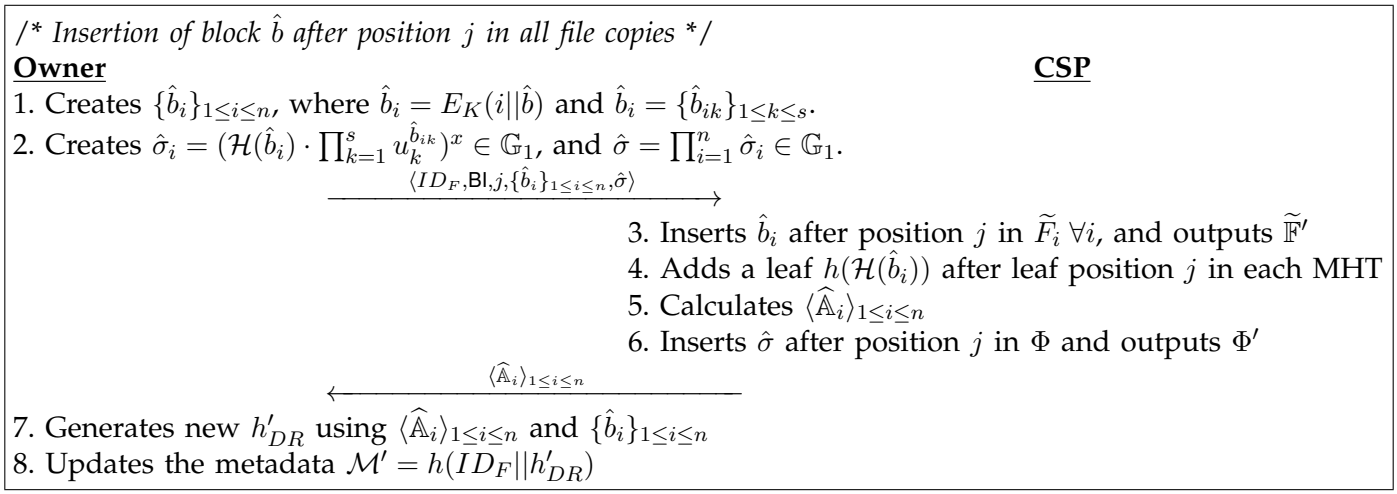


Fig. 8: The block insertion protocol in the TB-DMCPDP scheme

- **Deletion**. Block deletion operation is the opposite of the insertion operation. When one block is deleted all subsequent blocks are moved one step forward. To delete a specific data block at position j from all copies, the owner sends a delete request $\langle ID_F, \mathsf{BD}, j, null, null\rangle$ to the CSP. Upon receiving this request, the CSP runs the ExecUpdate algorithms to do the following:
  1) Calculates the authentication paths $\langle\mathbb{A}_{ij}\rangle_{1\le i\le n}$ of the blocks at position j (blocks to be deleted) in all copies. $\mathbb{A}_{ij}$ is an *ordered* set of node siblings on the path from the leaf node $h(\mathcal{H}(\tilde{b}_{ij}))$ to the root of the MHT of copy i
  2) Deletes the existing blocks $\{\tilde{b}_{ij}\}_{1\le i\le n}$ and the leaf nodes $\{h(\mathcal{H}(\tilde{b}_{ij}))\}_{1\le i\le n}$, and outputs new file copies $\widetilde{\mathbb{F}}'$ along with a new version of the MHTs
  3) Deletes σ_j from Φ, and outputs $\Phi' = \{\sigma_1, \sigma_2 \ldots, \sigma_{j-1}, \sigma_{j+1} \ldots, \sigma_{m-1}\}$
  4) Sends $\langle\mathbb{A}_{ij}\rangle_{1\le i\le n}$ to the owner

  The owner uses the authentication information received from the CSP to generate a new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$.

  Remark 2. Appendix A contains examples that demonstrate how the dynamic operations performed on the outsourced file copies affect the MHTs on the CSP side. Moreover, these examples show how the owner uses the information received from the CSP to generate the new directory root and update the metadata.

- ***Challenge***. For challenging the CSP and validating the integrity and consistency of all copies, the verifier sends c (# of blocks to be challenged) and two fresh keys at each challenge: a PRP(π) key k_1 and a PRF(ψ) key k_2. *Both* the verifier and the CSP use the PRP(π) keyed with k_1 and the PRF(ψ) keyed with k_2 to generate a set $Q = \{(j, r_j)\}$ of pairs of random indices and random values, where $\{j\} = \pi_{k_1}(l)_{1\le l\le c}$ and $\{r_j\} = \psi_{k_2}(l)_{1\le l\le c}$.
- ***Response***. After generating the set $Q = \{(j, r_j)\}$ of random indices and values, the CSP runs the Prove algorithm to generate an evidence that it is still correctly possessing the n copies in an updated and consistent state. The CSP responds with a proof $\mathbb{P} = \langle\sigma, \mu, \{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1\le i\le n\\(j,*)\in Q}}, \langle\mathbb{A}_{ij}\rangle_{\substack{1\le i\le n\\(j,*)\in Q}}\rangle$, where

$$\sigma = \prod_{(j,r_j)\in Q}\sigma_j^{r_j} \in \mathbb{G}_1\ , \qquad \mu_{ik} = \sum_{(j,r_j)\in Q} r_j\cdot\tilde{b}_{ijk} \in \mathbb{Z}_p, \qquad \text{and } \mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\1\le k\le s}}.$$

  $\langle\mathbb{A}_{ij}\rangle_{\substack{1\le i\le n\\(j,*)\in Q}}$ are the authentication paths of $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1\le i\le n\\(j,*)\in Q}}$.
- ***Verify Response***. Upon receiving the proof $\mathbb{P} = \langle\sigma, \mu, \{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1\le i\le n\\(j,*)\in Q}}, \langle\mathbb{A}_{ij}\rangle_{\substack{1\le i\le n\\(j,*)\in Q}}\rangle$ from the CSP, the verifier runs the Verify algorithm to do the following:

1) Constructs the directory root h_{DR} using $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1\le i\le n\\(j,*)\in Q}}$ and $\langle \mathbb{A}_{ij}\rangle_{\substack{1\le i\le n\\(j,*)\in Q}}$
2) Generates the value $\mathcal{V} = h(ID_F||h_{DR})$, and checks $\mathcal{V} \stackrel{?}{=} \mathcal{M}$, where $\mathcal{M}$ is the authenticated-most-recent metadata stored on the verifier side. If the checking fails, returns 0, otherwise the Verify algorithm checks following verification equation:

$$\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}(\prod_{i=1}^{n} \prod_{(j,r_j)\in Q} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y) \quad (1)$$

If the verification equation passes, the Verify algorithm returns 1, otherwise 0. The correctness of the above verification equation (1) can be shown as follows:

$$\begin{aligned}
\hat{e}(\sigma, g) &= \hat{e}(\prod_{(j,r_j)\in Q} \sigma_j^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} \sigma_{ij}]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(\tilde{b}_{ij}) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^{x}]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(\tilde{b}_{ij}) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})]^{r_j}, y) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \prod_{k=1}^{s} u_k^{r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}(\prod_{i=1}^{n} \prod_{(j,r_j)\in Q} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}(\prod_{i=1}^{n} \prod_{(j,r_j)\in Q} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y)
\end{aligned}$$

The TB-DMCPDP scheme can be slightly modified to minimizing the communication overhead by a factor of n via allowing the CSP to compute and send $\mu = \{\hat{\mu}_k\}_{1\le k\le s}$, where $\hat{\mu}_k = \sum_{i=1}^{n} \mu_{ik}$. However, this modification enables the CSP to simply cheat the verifier as follows:

$$\hat{\mu}_k = \sum_{i=1}^{n} \mu_{ik} = \sum_{i=1}^{n} \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk} = \sum_{(j,r_j)\in Q} r_j \cdot \sum_{i=1}^{n} \tilde{b}_{ijk}$$

Thus, the CSP can just keep the sectors summation $\sum_{i=1}^{n} \tilde{b}_{ijk}$ not the sectors themselves. Moreover, the CSP can corrupt the block sectors and the summation is still valid. Therefore, we force the CSP to send $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\1\le k\le s}}$, and the summation $\sum_{i=1}^{n} \mu_{ik}$ is done on the verifier side. The challenge response protocol in the TB-DMCPDP scheme is summarized in Fig. 9.

3.6 The Proposed MB-DMCPDP Scheme

3.6.1 Map-Version Table

Our proposed MB-DMCPDP scheme is based on using a map-version table to support outsourcing of dynamic data. This table is a small data structure stored on the verifier side to validate the integrity and consistency of all file copies stored by the CSP. The map-version table consists of three columns: serial number ($\mathcal{SN}$), block number ($\mathcal{BN}$), and version number ($\mathcal{VN}$). The $\mathcal{SN}$ is an indexing to the file blocks. It indicates the *physical* position of a block in a data file. The $\mathcal{BN}$ is a counter used to make a *logical* numbering/indexing to the file blocks. Thus, the relation between $\mathcal{BN}$ and $\mathcal{SN}$ can be viewed as

Verifier | **CSP**

1. Picks c (# of blocks to be challenged) and two fresh keys k_1 and k_2
2. Generates a set $Q = \{(j, r_j)\}$: $\{j\} = \pi_{k_1}(l)_{1 \le l \le c}$ and $\{r_j\} = \psi_{k_2}(l)_{1 \le l \le c}$

$$\xrightarrow[c,\, k_1,\, k_2]{}$$

3. Generates a set Q as the verifier did
4. Computes $\sigma = \prod_{(j,r_j) \in Q} \sigma_j^{r_j} \in \mathbb{G}_1$
5. Computes $\mu = \{\mu_{ik}\}$, $\mu_{ik} = \sum_{(j,r_j) \in Q} r_j \cdot \tilde{b}_{ijk} \in \mathbb{Z}_p$ $(i : 1 \to n,\ k : 1 \to s)$
6. Calculates $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n \\ (j,*) \in Q}}$ and $\langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n \\ (j,*) \in Q}}$

$$\xleftarrow[\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n \\ (j,*) \in Q}},\ \langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n \\ (j,*) \in Q}}]{\sigma, \mu,}$$

7. Constructs h_{DR} using $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n \\ (j,*) \in Q}}$ and $\langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n \\ (j,*) \in Q}}$
8. Computes $\mathcal{V} = h(ID_F || h_{DR})$, and checks $\mathcal{V} \stackrel{?}{=} \mathcal{M}$ (if fails returns 0).
9. Checks $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}(\prod_{i=1}^{n} \prod_{(j,r_j) \in Q} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y)$

Fig. 9: The challenge response protocol in the TB-DMCPDP scheme

a mapping between the logical number $\mathcal{BN}$ and the physical position $\mathcal{SN}$. The $\mathcal{VN}$ indicates the current version of file blocks. When a data file is initially created the $\mathcal{VN}$ of all blocks is 1. If a specific block is being updated, its $\mathcal{VN}$ is incremented by 1. It is of significant importance to note that the verifier keeps only *one* table for any arbitrary number of file copies.

3.6.2 MB-DMCPDP Procedural Steps

The procedures of our MB-DMCPDP scheme execution are as follows:

- ***Key Generation***. The same as in the TB-DMCPDP scheme.
- ***Generation of Distinct Copies***. The same as in the TB-DMCPDP scheme.
- ***Generation of Tags***. Given the distinct file copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}$, where $\widetilde{F}_i = \{\tilde{b}_{ijk}\}$, the data owner runs the TagGen algorithm to create a tag σ_{ij} for each block $\tilde{b}_{ij}$ as $\sigma_{ij} = (\mathcal{H}(ID_F || \mathcal{BN}_j || \mathcal{VN}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x \in \mathbb{G}_1$ $(i : 1 \to n,\ j : 1 \to m,\ k : 1 \to s)$. In the tag computation, $\mathcal{BN}_j$ is the logical number of the block at *physical* position j, $\mathcal{VN}_j$ is the current version of that block, and ID_F is a unique file identifier for each owner's file constructed as $Filename||n||u_1||\ldots||u_s$. The ID_F is embedded into the block tag to prevent the CSP from cheating using blocks from different files. The data owner then generates an aggregated tag σ_j for the blocks at the same indices in each copy $\widetilde{F}_i$ as $\sigma_j = \prod_{i=1}^{n} \sigma_{ij} \in \mathbb{G}_1$. Let us denote the set of aggregated tags as $\Phi = \{\sigma_j\}_{1 \le j \le m}$. The data owner sends $\{\mathbb{F}, \Phi, ID_F\}$ to the CSP, and deletes the copies and the tags from its local storage. The map-version table is stored on the local storage of the owner (or any trusted verifier).
- ***Dynamic Operations on the Data Copies***. The dynamic operations in the proposed MB-DMCPDP scheme are performed at the block level — as in our TB-DMCPDP scheme — via a request in the general form $\langle ID_F, BlockOp, j, \{b_i^*\}_{1 \le i \le n}, \sigma_j^* \rangle$. ID_F is the file identifier, *BlockOp* is BM for modification, BI for insertion, or BD for delete, j indicates the index of the block to be updated, $\{b_i^*\}_{1 \le i \le n}$ are the new block values for all copies, and σ_j^* is the new aggregated tag for the new blocks.
  - **Modification**. For a file $F = \{b_1, b_2, \ldots, b_m\}$, suppose the owner wants to modify the block b_j with the block b'_j for all file copies $\widetilde{\mathbb{F}}$. The owner runs the PrepareUpdate algorithm to do the

following:

1) Updates $\mathcal{VN}_j = \mathcal{VN}_j + 1$ in the map-version table
2) Creates n distinct blocks $\{\tilde{b}'_{ij}\}_{1\leq i\leq n}$, where $\tilde{b}'_{ij} = E_K(i||b'_j)$ is fragmented into s sectors $\{\tilde{b}'_{ij1}, \tilde{b}'_{ij2}, \ldots, \tilde{b}'_{ijs}\}$
3) Creates a new tag σ'_{ij} for each block $\tilde{b}'_{ij}$ as $\sigma'_{ij} = (\mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)\cdot\prod_{k=1}^{s} u_k^{\tilde{b}'_{ijk}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\sigma'_j = \prod_{i=1}^{n}\sigma'_{ij} \in \mathbb{G}_1$
4) Sends a modify request $\langle ID_F, \mathsf{BM}, j, \{\tilde{b}'_{ij}\}_{1\leq i\leq n}, \sigma'_j\rangle$ to the CSP

Upon receiving the modify request from the data owner, the CSP runs the ExecUpdate algorithms to do the following:

1) Replaces the block $\tilde{b}_{ij}$ with $\tilde{b}'_{ij} \forall i$, and constructs an updated version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1\leq i\leq n}$
2) Replaces σ_j with σ'_j in the set Φ, and outputs $\Phi' = \{\sigma_1, \sigma_2, \ldots, \sigma'_j, \ldots, \sigma_m\}$

- **Insertion**. For a file $F = \{b_1, b_2, \ldots, b_m\}$, suppose the owner wants to insert a new block $\hat{b}$ *after* position j for all file copies $\widetilde{\mathbb{F}}$. The data owner runs the PrepareUpdate algorithm to do the following:
  1) Constructs a new table entry $\langle \mathcal{SN}, \mathcal{BN}, \mathcal{VN}\rangle$ = $\langle j$+1, $(Max\{\mathcal{BN}_j\}_{1\leq j\leq m})$+1, 1 $\rangle$, and inserts this entry in the map-version table after position j
  2) Creates n distinct blocks $\{\hat{b}_i\}_{1\leq i\leq n}$, where $\hat{b}_i = E_K(i||\hat{b})$ is fragmented into s sectors $\{\hat{b}_{i1}, \hat{b}_{i2}, \ldots, \hat{b}_{is}\}$
  3) Creates a new tag $\hat{\sigma}_i$ for each block $\hat{b}_i$ as $\hat{\sigma}_i = (\mathcal{H}(ID_F||\mathcal{BN}_{j+1}||\mathcal{VN}_{j+1}) \cdot \prod_{k=1}^{s} u_k^{\hat{b}_{ik}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\hat{\sigma} = \prod_{i=1}^{n}\hat{\sigma}_i \in \mathbb{G}_1$. Note that $\mathcal{BN}_{j+1}$ is the logical number of the new block with current version $\mathcal{VN}_{j+1}$ = 1
  4) Sends an insert request $\langle ID_F, \mathsf{BI}, j, \{\hat{b}_i\}_{1\leq i\leq n}, \hat{\sigma}\rangle$ to the CSP

  Upon receiving the insert request from the owner, the CSP runs the ExecUpdate algorithms to do the following:

  1) Inserts the block $\hat{b}_i$ after position j in the file copy $\widetilde{F}_i$ $\forall i$, and constructs a new version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1\leq i\leq n}$
  2) Inserts $\hat{\sigma}$ after position j in the set Φ, and outputs $\Phi' = \{\sigma_1, \sigma_2, \ldots, \sigma_j, \hat{\sigma}, \ldots, \sigma_{m+1}\}$, i.e., $\sigma_{j+1} = \hat{\sigma}$
- **Append**. Block append operation means adding a new block at the end of the outsourced data. It can simply be implemented via insert operation after the last block of the data file.
- **Deletion**. To delete a specific data block at position j from all copies, the owner deletes the entry at position j from the map-version table and sends a delete request $\langle ID_F, \mathsf{BD}, j, null, null\rangle$ to the CSP. Upon receiving this request, the CSP runs the ExecUpdate algorithms to do the following:
  1) Deletes the existing blocks $\{\tilde{b}_{ij}\}_{1\leq i\leq n}$, and outputs a new version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}_i\}_{1\leq i\leq n}$
  2) Deletes σ_j from Φ and outputs $\Phi' = \{\sigma_1, \sigma_2, \ldots, \sigma_{j\text{-}1}, \sigma_{j+1}, \ldots, \sigma_{m-1}\}$.

Fig. 10 shows the changes in the map-version table due to dynamic operations on a file $F = \{b_j\}_{1\leq j\leq 8}$. When the file is initially created $\mathcal{SN}_j = \mathcal{BN}_j$, and $\mathcal{VN}_j$ = 1: $1 \leq j \leq 8$. Fig. 10.b shows that $\mathcal{VN}_5$ is incremented by 1 for updating the block at position 5. To insert a new block after position 3, Fig. 10.c shows that a new entry $\langle 4, 9, 1\rangle$ is inserted in the map-version table after $\mathcal{SN}_3$, where 4 is the physical position of the newly inserted block, 9 is the new logical block number computed by incrementing the maximum of all previous logical block numbers, and 1 is the version of the new block. Deleting a block at position 2 requires deleting the table entry at $\mathcal{SN}_2$ and shifting all subsequent entries one position up (Fig. 10.d). Note that during all dynamic operations, the $\mathcal{SN}$ indicates the actual physical positions of the data blocks in the file copies $\widetilde{\mathbb{F}}$.

- ***Challenge***. The same as in the TB-DMCPDP scheme.
- ***Response***. After generating the set $Q = \{(j, r_j)\}$ of random indices and values, the CSP runs the Prove

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{VN}$
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1

(a)Intially $\Longrightarrow$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{VN}$
1	1	1
2	2	1
3	3	1
4	4	1
5	***5***	***2***
6	6	1
7	7	1
8	8	1

(b)Modifying block at position 5 $\Longrightarrow$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{VN}$
1	1	1
2	2	1
3	3	1
4	***9***	***1***
5	4	1
6	5	2
7	6	1
8	7	1
9	8	1

(c)Insert block after position 3 $\Longrightarrow$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{VN}$
1	1	1
2	3	1
3	9	1
4	4	1
5	5	2
6	6	1
7	7	1
8	8	1

(d)After deleting at position 2

Fig. 10: Changes in the map-version table due to different dynamic operations.

algorithm to generate an evidence that it is still correctly possessing the n copies in an updated and consistent state. The CSP responds with a proof $\mathbb{P} = \{\sigma, \mu\}$, where

$$\sigma = \prod_{(j,r_j)\in Q} \sigma_j^{r_j} \in \mathbb{G}_1 \ , \qquad \mu_{ik} = \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk} \in \mathbb{Z}_p, \qquad \text{and } \mu = \{\mu_{ik}\}_{\substack{1\le i\le n \\ 1\le k\le s}}$$

- ***Verify Response***. Upon receiving the proof $\mathbb{P} = \{\sigma, \mu\}$ from the CSP, the verifier runs the Verify algorithm to check the following verification equation:

$$\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y) \tag{2}$$

If the verification equation passes, the Verify algorithm returns 1, otherwise 0. The correctness of the above verification equation (2) can be determined as follows:

$$\begin{aligned}
\hat{e}(\sigma, g) &= \hat{e}(\prod_{(j,r_j)\in Q} \sigma_j^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} \sigma_{ij}]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})]^{r_j}, y) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j} \cdot \prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \prod_{k=1}^{s} u_k^{r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y)
\end{aligned}$$

We have not let the CSP to compute and send $\mu = \{\hat{\mu}_k\}_{1\le k\le s}$, where $\hat{\mu}_k = \sum_{i=1}^{n} \mu_{ik}$ as a countermeasure against a cheating that may be done on the CSP side by keeping only the sectors summation $\sum_{i=1}^{n} \tilde{b}_{ijk}$, not the sectors themselves, as we have previously illustrated in the TB-DMCPDP scheme. The challenge response protocol in the MB-DMCPDP scheme is summarized in Fig. 11.

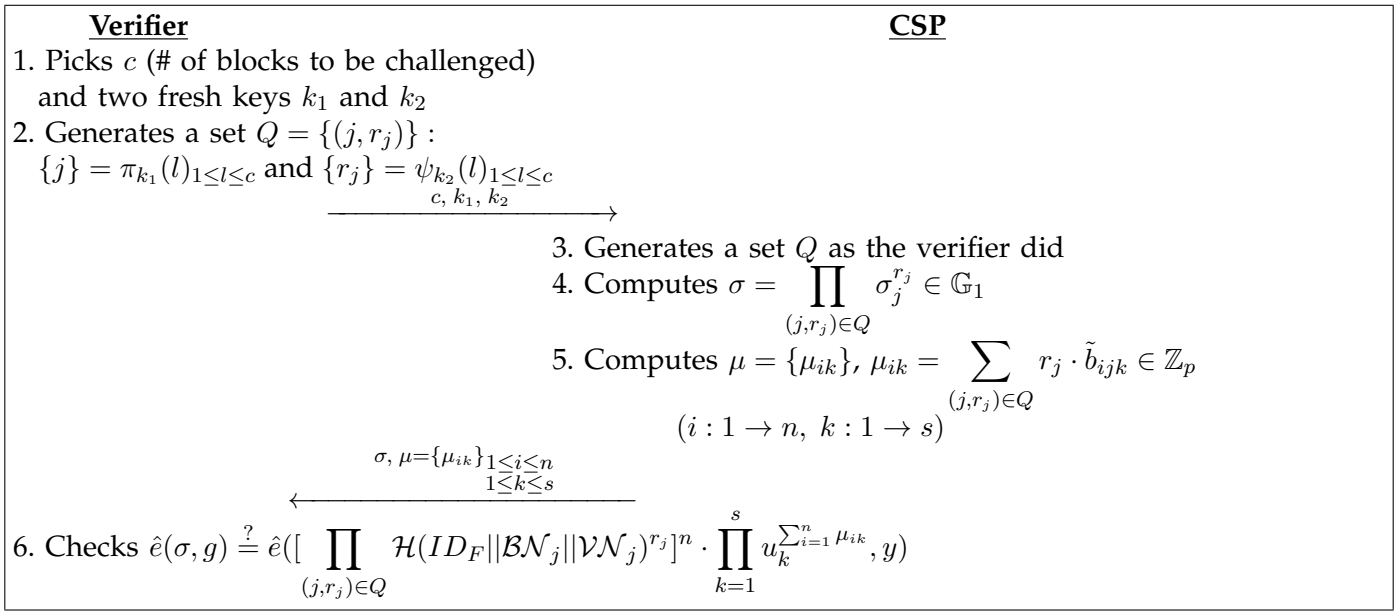


Fig. 11: The challenge response protocol in the MB-DMCPDP scheme

Remark 2. The proposed two schemes in this work support public verifiability where anyone (not necessarily the owner) who knows the owner's public key can send a challenge vector to the CSP and verify the response. Public verifiability can solve disputes that may occur between the data owner and the CSP regarding data integrity. If such a dispute occurs, we can resort to a trusted TPA to determine whether the data integrity is maintained or not. Since the owner's public key is only needed to perform the verification protocol, the owner is not required to reveal his secret key to the TPA. The formal security analysis of the proposed schemes is presented in Appendix B.

4 Performance Analysis and Experimental Results

4.1 Performance Analysis

In this section we evaluate the performance of the proposed schemes: TB-DMCPDP and MB-DMCPDP. The computation cost of these schemes is estimated in terms of the cryptographic operations notated in Table 1. $\mathbb{G}_1$ indicates a group of points over a suitable elliptic curve in the bilinear pairing.

TABLE 1: Notation of cryptographic operations

Notation	Description
h_{SHA}	Cryptographic hashing
$\mathcal{H}_{\mathbb{G}_1}$	Hashing to $\mathbb{G}_1$
$\mathcal{E}_{\mathbb{G}_1}$	Exponentiation in $\mathbb{G}_1$
$\mathcal{M}_{\mathbb{G}_1}$	Multiplication in $\mathbb{G}_1$
$\mathcal{M}_{\mathbb{Z}_p}$	Multiplication in $\mathbb{Z}_p$
$\mathcal{A}_{\mathbb{Z}_p}$	Addition in $\mathbb{Z}_p$
$\mathcal{P}$	Bilinear pairing

Let n, m, and s denote the number of copies, the number of blocks per copy, and the number of sectors per block, respectively. Let c denotes the number of blocks to be challenged, and $|F|$ denotes the size of the file copy. Without loss of generality, we assume that the desired security level is 128-bit. Thus, we utilize an elliptic curve defined over Galois field $GF(p)$ with $|p|$ = 256 bits (a point on this curve can be represented by 257 bits using compressed representation [5]), and a cryptographic hash of size 256 bits (e.g., SHA-256). Let the keys used with the PRP and the PRF be of size 128 bits. Table 2 presents a theoretical analysis for the setup, storage, communication, computation, and dynamic operations costs of the proposed schemes.

TABLE 2: Performance of the proposed schemes. ‡ $\log_2(m)$ is the upper bound of the authentication path length when $c > 1$

		TB-DMCPDP	MB-DMCPDP
System Setup	Tag Generation	$(s+1)mn\,\mathcal{E}_{\mathbb{G}_1}$+ $mn\,\mathcal{H}_{\mathbb{G}_1}$ +$(sn+n-1)m\,\mathcal{M}_{\mathbb{G}_1}$	$(s+1)mn\,\mathcal{E}_{\mathbb{G}_1}$+ $mn\,\mathcal{H}_{\mathbb{G}_1}$ +$(sn+n-1)m\,\mathcal{M}_{\mathbb{G}_1}$
	Metadata Generation	$mn\,\mathcal{H}_{\mathbb{G}_1} + (2m+1)n\,h_{SHA}$	—
Storage	File Copies	$n\|F\|$	$n\|F\|$
	CSP Overhead	(257+512n)m bits	257 m bits
	Verifier Overhead	256 bits	64 m bits
Communication Cost	Challenge	256+$\log_2(c)$ *bits*	256+$\log_2(c)$ *bits*
	Response	257 + 256sn + (256 $\log_2(m)$+257)cn bits ‡	257+256sn bits
Computation Cost	Proof	$c\,\mathcal{E}_{\mathbb{G}_1}$+$(c-1)\,\mathcal{M}_{\mathbb{G}_1}$+$csn\,\mathcal{M}_{\mathbb{Z}_p}$ +$(c-1)sn\,\mathcal{A}_{\mathbb{Z}_p}$+$cn\,\mathcal{H}_{\mathbb{G}_1}$	$c\,\mathcal{E}_{\mathbb{G}_1}$+$(c-1)\,\mathcal{M}_{\mathbb{G}_1}$+$csn\,\mathcal{M}_{\mathbb{Z}_p}$ +$(c-1)sn\,\mathcal{A}_{\mathbb{Z}_p}$
	Verify	$(c\,\log_2(m)+2)n\,h_{SHA}$‡ +2$\mathcal{P}$+$(c+s)\,\mathcal{E}_{\mathbb{G}_1}$+$cn\,\mathcal{H}_{\mathbb{G}_1}$ +$(cn+s-1)\,\mathcal{M}_{\mathbb{G}_1}$+$s(n-1)\,\mathcal{A}_{\mathbb{Z}_p}$	2$\mathcal{P}$+$(c+s+1)\,\mathcal{E}_{\mathbb{G}_1}$+ $c\,\mathcal{H}_{\mathbb{G}_1}$ +$(c+s-1)\,\mathcal{M}_{\mathbb{G}_1}$+$s(n-1)\mathcal{A}_{\mathbb{Z}_p}$
Dynamic Operations	Communication	"Request" + $O(n\log_2(m))$	"Request"
	State update	$O(n\log_2(m))\,h_{SHA}$	—

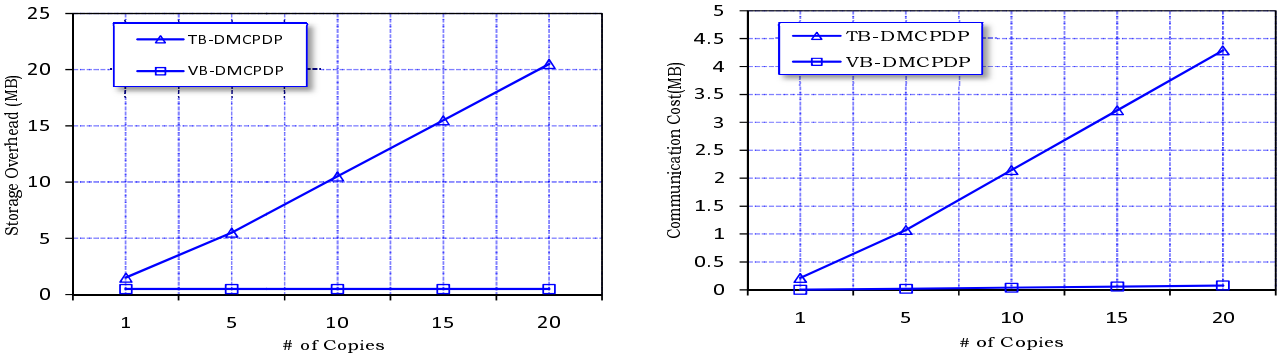
4.2 Comments

Table 2 shows that the setup cost of the TB-DMCPDP scheme is higher than that of the MB-DMCPDP scheme. This is due to the preparation of the MHTs for the file copies to generate the metadata $\mathcal{M}$. The storage overhead on the CSP for the MB-DMCPDP scheme is much less than that of the TB-DMCPDP model. Storage overhead is the additional space used to store some information other than the outsourced file copies $\widetilde{\mathbb{F}}$ ($n|F|$ bits are used to store $\widetilde{\mathbb{F}}$). Both schemes need some additional space to store the aggregated block tags $\Phi = \{\sigma_j\}_{1\leq j\leq m}$, where σ_j is an element in $\mathbb{G}_1$ that can be represented by 257 bits. Besides Φ, the TB-DMCPDP scheme needs to store a MHT for each file copy which costs additional storage space on the cloud servers. The MHTs can be computed on the fly during the operations of the TB-DMCPDP scheme. This slight modification can reduce the storage overhead on the remote servers, but it will negatively affect the overall system performance. The MHTs are needed through each dynamic operation of the file blocks and through the verification phase of the system. Thus, being not explicitly stored on the CSP can influence the system performance.

On the other hand, the TB-DMCPDP scheme only keeps a small metadata on the verifier side compared to a map-version table with the MB-DMCPDP scheme. An entry of the map-version table is of size 8 bytes (two integers), and the total number of entries equals to the number of file blocks. It is important to note that during implementation the $\mathcal{SN}$ is not needed to be stored in the table; $\mathcal{SN}$ is considered to be the entry/table index (the map-version table is implemented as a linked list). Moreover, there is only *one* table for all file copies which mitigates the storage overhead on the verifier side. The storage complexities are $O(m)$ and $O(mn)$ for the map-version table and the n copies, respectively. For example, if we are dealing with 64MB file with block size 4KB, the size of the map-version table will be only 128KB for any *arbitrary number* of copies (table size = 2MB for 1GB file).

From Table 2, the communication cost of the MB-DMCPDP scheme is less than that of the TB-DMCPDP scheme. During the response phase, the map-based scheme sends one element σ (257 bits) and $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}$, where μ_{ik} is represented by 256 bits. On the other hand, the tree-based approach sends $\langle \sigma, \mu, \{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1\leq i\leq n\\ (j,*)\in Q}}, \langle \mathbb{A}_{ij}\rangle_{\substack{1\leq i\leq n\\ (j,*)\in Q}}\rangle$, where each $\mathcal{H}(\tilde{b}_{ij})$ is represented by 257 bits, and $\mathbb{A}_{ij}$ is an authentication path of length $\approx O(\log_2 m)$. Each node along $\mathbb{A}_{ij}$ is a cryptographic hash of size 256 bits.

The CSP storage overhead and the communication cost during the *response* phase of the proposed schemes are depicted in Fig. 12 for 64MB file with 4KB block size. Fig. 12.a shows the storage overhead in MB on the remote servers for different number of copies. For the MB-DMCPDP scheme, the storage overhead is independent of the number of copies n, while it is linear in n for the TB-DMCPDP scheme. For 20 copies, the overhead on the CSP is 20.50MB and 0.50MB for the TB-DMCPDP and MB-DMCPDP



(a) CSP storage overhead (MB) **(b)** Communication cost (MB)

Fig. 12: Storage and communication overhead of the proposed schemes.

schemes, respectively. Fig. 12.b shows the communication cost for both schemes using different number of copies. The communication cost of the MB-DMCPDP scheme is much lower than that of the TB-DMCPDP scheme. The response of the TB-DMCPDP scheme for 20 copies is 4.29MB, while it is 0.078MB for the MB-DMCPDP scheme (the response is reduced by 98%). The challenge for both schemes is about 34 bytes.

Table 2 also presents the cost of dynamic operations for both schemes. During the dynamic operations of the TB-DMCPDP scheme, the owner sends a request $\langle ID_F, BlockOp, j, \{b_i^*\}_{1\leq i\leq n}, \sigma_j^*\rangle$ to the CSP and receives the authentication paths. Updating the state — MHTs on the CSP and $\mathcal{M}$ on the owner — costs $O(n\log_2(m))\, h_{SHA}$ to update the MHTs of the file copies according to the required dynamic operations, and regenerate the new directory root that constructs a new $\mathcal{M}$. On the other hand, for the MB-DMCPDP scheme, the owner only sends a request to the CSP and updates the map-version table (the state) without usage of cryptographic operations (add, remove, or modify a table entry).

4.3 Experimental Results

The experiments are conducted using C++ on a system with an Intel(R) Xeon (R) 2-GHZ processor and 3 GB RAM running Windows XP. In our implementation we use MIRACL library version 5.4.2 and 64MB file with block size 4KB. For 128-bit security level, the elliptic curve group we work on has a 256-bit group order. In the experiments, we utilize the Barreto-Naehrig(BN) [4] curve defined over prime field $GF(p)$ with $|p|$ = 256 and embedding degree = 12 (the BN curve with these parameters is provided by the MIRACL library). In the implementation we do not consider the time to access the file blocks, as the state-of-the-art hard drive technology allows as much as 1MB to be read in just few nanoseconds [28]. Hence, the total access time is unlikely to have substantial impact on the overall system performance. Table 3 presents the timing measurements (in milliseconds) obtained from the MIRACL library for the cryptographic operations used in the implementations. All measurements are averages of thousands of runs. The pairing operation consumes more time than other cryptographic operations, but the proposed schemes use just only two pairings in the verification phase.

TABLE 3: Timing measurements for the cryptographic operations

Operation	h_{SHA}	$\mathcal{H}_{\mathbb{G}_1}$	$\mathcal{E}_{\mathbb{G}_1}$	$\mathcal{M}_{\mathbb{G}_1}$	$\mathcal{M}_{\mathbb{Z}_p}$	$\mathcal{A}_{\mathbb{Z}_p}$	$\mathcal{P}$
Time(ms)	0.012937	0.4728	2.0047	0.01156	0.00043	0.00015	88.656

We compare the proposed two schemes from both the CSP computation cost and the verifier computation cost. It has been reported in [2] that if the remote server is missing a fraction of the data, then the number of blocks that needs to be checked in order to detect server misbehavior with high probability is constant independent of the total number of file blocks. For example, if the server deletes 1% of the data

file, the verifier only needs to check for c = 460-randomly chosen blocks of the file so as to detect this misbehavior with probability larger than 99%. Therefore, in our experiments, we use c = 460 to achieve a high probability of assurance.

For different number of copies, Table 4 presents the CSP computation times (in seconds) to provide an evidence that the file copies are actually stored on the cloud servers in an updated, uncorrupted, and consistent state. Also, it presents the verifier computation times (in seconds) to check the responses received from the CSP. The *speed-ups* S_{Compt} and S_{Verify} of the MB-DMCPDP with respect to the TB-DMCPDP are also presented. The *speed-up* = $\frac{Time_{\text{TB-DMCPDP}} - Time_{\text{MB-DMCPDP}}}{Time_{\text{TB-DMCPDP}}} \times 100$.

TABLE 4: CSP computation times (sec) and verifier computation times (sec) of the proposed schemes for different number of copies.

	CSP Computation Times		Verifier Computation Times		Speed-ups	
# of Copies	TB-DMCPDP	MB-DMCPDP	TB-DMCPDP	MB-DMCPDP	S_{Compt}	S_{Verify}
1	1.17909	0.9616	1.66369	1.58035	18.45%	4.89%
5	2.18556	1.09812	2.88835	1.58243	49.76%	45.21%
10	3.44366	1.26878	4.41917	1.58253	63.16%	64.19%
15	4.70176	1.43944	5.95	1.58262	69.39%	73.4%
20	5.95985	1.61009	7.48082	1.58272	72.98%	78.84%
50	13.50843	2.63403	16.66577	1.58329	80.5%	90.5%
100	26.08939	4.34059	31.97402	1.58425	83.36%	95.05%

Table 4 shows that the map-based scheme outperforms the tree-based scheme from both the CSP and the verifier computation times. The computation time over the cloud servers can be reduced up to 83%, and the verification time can be reduced up to 95%. As illustrated in Table 4, the MB-DMCPDP scheme has a very small increase in the verification time with increasing number of copies. This is due to the fact that although the term $s(n-1)\mathcal{A}_{\mathbb{Z}_p}$ in the verification cost of the MB-DMCPDP scheme is linear in n (§ Table 2), in our experiments its numerical value is quite small compared to those of the other terms in the cost expression. This feature makes the map-based scheme more efficient when verifying a large number of file copies, and computationally cost-effective when the verification process is done in a constrained environment with limited verifier's computational power.

5 Identifying Corrupted Copies

Here, we show how the proposed MB-DMCPDP scheme can be slightly modified to identify the indices of corrupted copies. The same ideas can be applied to the TB-DMCPDP scheme.

The proof $\mathbb{P} = \{\sigma, \mu\}$ generated by the CSP will be valid and will pass the verification equation (2) only if all copies are intact and consistent. Thus, when there is one or more corrupted copies, the whole auditing procedure fails. To handle this situation and identify the corrupted copies, a slightly modified version of the MB-DMCPDP scheme can be used. In this version, the data owner generates a tag σ_{ij} for each block $\tilde{b}_{ij}$, but does not aggregate the tags for the blocks at the same indices in each copy, i.e., $\Phi = \{\sigma_{ij}\}_{\substack{1\le i\le n \\ 1\le j\le m}}$.

During the response phase, the CSP computes $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n \\ 1\le k\le s}}$ as before, but $\sigma = \prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} \sigma_{ij}]^{r_j} \in \mathbb{G}_1$.

Upon receiving the proof $\mathbb{P} = \{\sigma, \mu\}$, the verifier first validates $\mathbb{P}$ using equation (2). If the verification fails, the verifier asks the CSP to send $\sigma = \{\sigma_i\}_{1\le i\le n}$, where $\sigma_i = \prod_{(j,r_j)\in Q} \sigma_{ij}^{r_j}$. Thus, the verifier has two lists σList = $\{\sigma_i\}_{1\le i\le n}$ and μList = $\{\mu_{ik}\}_{\substack{1\le i\le n \\ 1\le k\le s}}$ (μList is a two dimensional list).

Utilizing a recursive divide-and-conquer approach (binary search) [14], the verifier can identify the indices of corrupted copies. Specifically, σList and μList are divided into halves: σList$\rightarrow$ (σLeft:σRight), and μList$\rightarrow$ (μLeft:μRight). The verification equation (2) is applied recursively on σLeft with μLeft and σRight with μRight. Note that the individual tags in σLeft or σRight are aggregated via multiplication to generate one σ that is used during the recursive application of equation (2). The procedural steps of identifying the indices of corrupted copies are indicated in Algorithm 1. The BS (binary search) algorithm

takes four parameters: σList, μList, start that indicates the start index of the currently working lists, and end to indicate the last index of the working lists. The initial call to the BS algorithm takes (σList, μList, 1, n). The invalid indices are stored in invalidList which is a global data structure.

Algorithm 1: BS(σList, μList, start, end)

begin
  len$\longleftarrow$ (end$-$start)$+1$ /* *The list length* */
  if *len = 1* **then**
    $\sigma \longleftarrow \sigma$List[start]
    $\{\mu_k\}_{1\leq k\leq s} \longleftarrow \mu$List[start][$k$]
    $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}(\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j} \cdot \prod_{k=1}^{s} u_k^{\mu_k}, y)$
    if *NOT verified* **then**
      invalidList.Add(start)
    end
  else
    $\sigma \longleftarrow \prod_{i=1}^{len} \sigma$List[start$+i-1$]
    $\{\mu_{ik}\}_{\substack{1\leq i\leq len \\ 1\leq k\leq s}} \longleftarrow \mu$List[start$+i-1$][$k$]
    $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^{len} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{len}\mu_{ik}}, y)$
    if *NOT verified* **then**
      /* *work with the left and right halves of* σList and μList */
      mid$\longleftarrow \lceil$(start+end)$/2\rceil$ /* *List middle* */
      BS(σList, μList, start, mid) /* *Left part* */
      BS(σList, μList, mid+1, end) /* *Right part* */
    end
  end
end

This slight modification to identify the corrupted copies increases the storage overhead on the cloud servers, where the CSP has to store mn tags for the file copies $\widetilde{\mathbb{F}}$ compared with m tags in the original version. Moreover, the challenge-response phase is done in two rounds if the initial round to verify all copies fails.

We design experiments to show the effect of identifying the corrupted copies on the verification time. We generate 100 copies and *randomly* corrupt a percentage — ranging from 1% to 50% — of the file copies. Table 5 presents the verification time (in seconds) with different corrupted percentages. The verification time is 18.99189 seconds when 1% of the copies are invalid compared with 1.58425 seconds (§Table 4) when all copies are intact. As observed from Table 5, if more than 15% of the copies are corrupted, it is more efficient to separately validate each file copy. Individual verifications of the 100 copies requires 100$\times$1.58035 = 158.035 seconds.

In short, the proposed schemes can be slightly modified to support the feature of identifying the corrupted copies at the cost of some extra storage, communication, and computation overheads.

6 Summary and Concluding Remarks

Increasingly more and more individuals and organizations are outsourcing their data to remote servers alleviating the burden of local data storage and maintenance. In this work we have studied the problem

TABLE 5: Verification times when different percentages of 100 copies are corrupted

Corrupted Percentage	Verification Time (sec)
1%	18.99189
5%	60.11692
10%	98.08061
15%	148.69183
20%	170.83106
50%	284.66757

of creating multiple copies of dynamic data file and verifying those copies stored on untrusted cloud servers.

We have proposed two schemes: TB-DMCPDP and MB-DMCPDP. These schemes support outsourcing of dynamic data, where the data owner is capable of not only archiving and accessing the data copies stored by the CSP, but also updating and scaling these copies on the remote servers. The interaction between the authorized users and the CSP is considered in our schemes, where the authorized users can seamlessly access a data copy received from the CSP using a single secret key shared with the data owner. Moreover, the proposed schemes support public verifiability, enable arbitrary number of auditing, and allow *possession-free* verification where the verifier has the ability to verify the data integrity even though he neither possesses nor retrieves the file blocks from the server.

The proposed schemes have been shown to be provably secure via security analysis. Through performance analysis and experimental results, we have demonstrated that the proposed MB-DMCPDP scheme outperforms the TB-DMCPDP approach that can be considered as a fine-tuned extension to the single-copy DPDP schemes based on authenticated data structures. Such extension leads to high storage overhead on the remote servers and high computations on both the CSP and the verifier sides. The dynamic block operations of the map-based approach are efficiently done with less communication and computation costs. Moreover, the MB-DMCPDP scheme significantly reduces the computation time during the challenge-response phase which makes it more efficient for practical applications where a large number of verifiers are connected to the CSP causing a huge computation overhead over the servers. Besides, it has lower storage overhead on the servers, and thus reduces the fees paid by the cloud customers.

A slight modification can be done on the proposed schemes to support the feature of identifying the indices of corrupted copies. The corrupted data copy can be reconstructed even from a complete damage using duplicated copies on other servers.

ACKNOWLEDGMENTS

This work was supported in part through an NSERC grant awarded to Dr. Hasan.

REFERENCES

[1] Amazon elastic compute cloud (Amazon EC2), http://aws.amazon.com/ec2/.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2007, pp. 598–609.

[3] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *SecureComm '08: Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, New York, NY, USA, 2008, pp. 1–10.

[4] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proceedings of SAC 2005, volume 3897 of LNCS*. Springer-Verlag, 2005, pp. 319–331.

[5] ——, "Ieee p1363.3 submission: Pairing-friendly elliptic curves of prime order with embedding degree 12," New Jersey: IEEE Standards Association, 2006.

[6] A. F. Barsoum and M. A. Hasan, "Provable possession and replication of data over cloud servers," Centre For Applied Cryptographic Research (CACR), University of Waterloo, Report 2010/32, 2010, http://www.cacr.math.uwaterloo.ca/techreports/2010/cacr2010-32.pdf.

[7] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, London, UK, 2001, pp. 514–532.

[8] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: theory and implementation," in *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security*. New York, NY, USA: ACM, 2009, pp. 43–54.

[9] R. Curtmola, O. Khan, and R. Burns, "Robust remote data checking," in *StorageSS '08: Proceedings of the 4th ACM international workshop on Storage security and survivability*. New York, NY, USA: ACM, 2008, pp. 63–68.

[10] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: multiple-replica provable data possession," in *28th IEEE ICDCS*, 2008, pp. 411–420.

[11] Y. Deswarte, J.-J. Quisquater, and A. Saïdane, "Remote integrity checking," in *6th Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, S. J. L. Strous, Ed., 2003, pp. 1–11.

[12] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *TCC '09: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 109–127.

[13] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2009, pp. 213–222.

[14] A. L. Ferrara, M. Green, S. Hohenberger, and M. Pedersen, "Practical short signature batch verification," in *The Cryptographer's Track at RSA Conference*, 2009, pp. 309–324.

[15] D. L. G. Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer," Cryptology ePrint Archive, Report 2006/150, 2006.

[16] N. Gohring, "Amazon's S3 down for several hours," Online at http://www.pcworld.com/businesscenter/article/142549/amazons_s3_down_for_severalhours.html, 2008.

[17] P. Golle, S. Jarecki, and I. Mironov, "Cryptographic primitives enforcing communication and storage complexity," in *FC'02: Proceedings of the 6th International Conference on Financial Cryptography*, Berlin, Heidelberg, 2003, pp. 120–135.

[18] Z. Hao and N. Yu, "A multiple-replica remote data possession checking protocol with public verifiability," in *Second International Symposium on Data, Privacy, and E-Commerce*, 2010.

[19] Z. Hao, S. Zhong, and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE Transactions on Knowledge and Data Engineering*, vol. 99, no. PrePrints, 2011.

[20] A. Juels and B. S. Kaliski, "PORs: Proofs of Retrievability for large files," in *CCS'07: Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 584–597.

[21] B. Krebs, "Payment processor breach may be largest ever," Online at http://voices.washingtonpost.com/securityfix/2009/01/payment_processor_breach_may_b.html, Jan. 2009.

[22] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, 2001.

[23] P. Mell and T. Grance, "Draft NIST working definition of cloud computing," Online at http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, 2009.

[24] A. Menezes, "An introduction to pairing-based cryptography," Lecture Notes 2005, Online at http://www.math.uwaterloo.ca/~ajmeneze/publications/pairings.pdf.

[25] R. C. Merkle, "Protocols for public key cryptosystems," *Security and Privacy, IEEE Symposium on*, vol. 0, p. 122, 1980.

[26] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *Trans. Storage*, vol. 2, no. 2, 2006.

[27] W. Pugh, "Skip lists: A probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, pp. 668–676, 1990.

[28] F. Sebé, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," *IEEE Trans. on Knowl. and Data Eng.*, vol. 20, no. 8, 2008.

[29] H. Shacham and B. Waters, "Compact proofs of retrievability," Cryptology ePrint Archive, Report 2008/073, 2008, http://eprint.iacr.org/.

[30] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to keep online storage services honest," in *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, Berkeley, CA, USA, 2007, pp. 1–6.

[31] M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," Cryptology ePrint Archive, Report 2008/186, 2008.

[32] C. E. Shannon, "Communication theory of secrecy systems," *Bell Syst.Tech. J.*, vol. 28, no. 4, 1949.

[33] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," Cryptology ePrint Archive, Report 2009/081, 2009, http://eprint.iacr.org/.

[34] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *ESORICS'09: Proceedings of the 14th European Conference on Research in Computer Security*, Berlin, Heidelberg, 2009, pp. 355–370.

[35] W. Wang, Z. Li, R. Owens, and B. Bhargava, "Secure and efficient access to outsourced data," in *CCSW '09: Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, New York,NY, USA, 2009, pp. 55–66.

[36] M. Xie, H. Wang, J. Yin, and X. Meng, "Integrity auditing of outsourced data," in *VLDB '07: Proceedings of the 33rd International Conference on Very Large Databases*, 2007, pp. 782–793.

[37] K. Zeng, "Publicly verifiable remote data integrity," in *Proceedings of the 10th International Conference on Information and Communications Security*, ser. ICICS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 419–434.

[38] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Efficient provable data possession for hybrid clouds," in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 756–758.

APPENDIX A
EXAMPLES OF THE TB-DMCPDP SCHEME

This appendix contains some examples of the TB-DMCPDP scheme that demonstrate the effect of dynamic operations on the MHTs over the CSP side. Moreover, these examples show how the owner uses the information received from the CSP to generate the new directory root and update the metadata. We assume that the data owner has a file of 4 blocks and the CSP stores n copies of this file. Also we assume that during the system setup, the owner and the CSP have agreed to use left-to-right sequence to generate the Merkle trees.

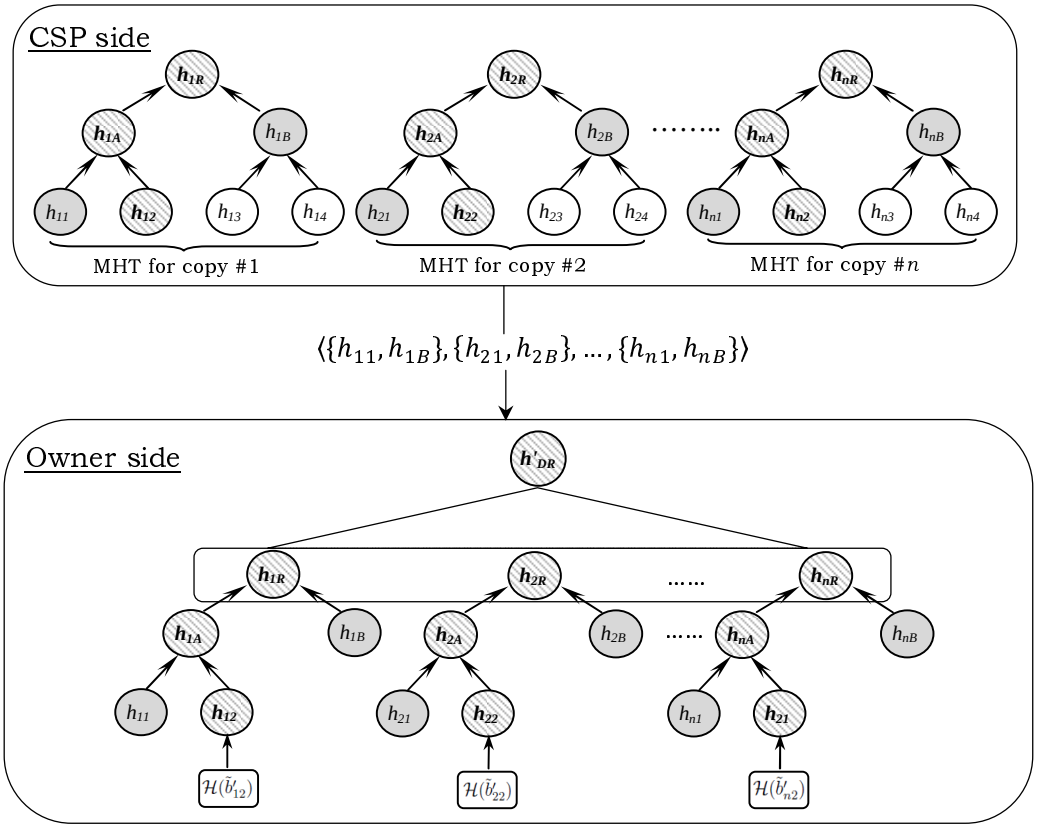


Fig. 13: Effect of block modification operation on the MHTs and the directory root.

- **Modification**. Fig. 13 shows that the second block is to be modified in all copies outsourced to the CSP. *On the CSP side*, the dashed nodes indicate the tree nodes that are updated due to the modification of the second block. The dashed *leaf* nodes $\{h_{12}, h_{22}, \ldots, h_{n2}\}$ are updated as $h_{i2} = h(\mathcal{H}(\tilde{b}'_{i2}))$, where $\tilde{b}'_{i2}$ — created and sent from the owner — is the modified second block dedicated for copy i: $1 \leq i \leq n$. The dashed *non-leaf* nodes are updated by h(left child $||$ right child). The grey nodes indicate the authentication paths of the modified blocks, e.g., $\{h_{11}, h_{1B}\}$ is the authentication path of the modified block $\tilde{b}'_{12}$. *On the owner side*, the owner uses the authentication paths $\langle\{h_{11}, h_{1B}\}, \{h_{21}, h_{2B}\}, \ldots, \{h_{n1}, h_{nB}\}\rangle$ sent from the CSP and the modified blocks $\{\tilde{b}'_{12}, \tilde{b}'_{22}, \ldots, \tilde{b}'_{n2}\}$ to generate the new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$. The dashed nodes indicate the generated cryptographic hashes using the authentication paths (grey circles) sent from the CSP and the modified blocks $\{\tilde{b}'_{i2}\}_{1\leq i\leq n}$. Thus, $h_{12} = h(\mathcal{H}(\tilde{b}'_{12}))$, $h_{1A} = h(h_{11}||h_{12})$, and $h_{1R} = h(h_{1A}||h_{1B})$. The computation of $\{h_{iR}\}_{2\leq i\leq n}$ is done in the same way. The owner uses the computed $\{h_{iR}\}_{1\leq i\leq n}$ to generate the updated directory root h'_{DR}, and finally computes the updated metadata $\mathcal{M}' = h(ID_F||h'_{DR})$.
- **Insertion**. Fig. 14 shows that a new block is to be inserted after position 2 in all copies outsourced to the CSP. *On the CSP side*, the cross-dashed nodes indicate the newly added leaf nodes, i.e., $\hat{h}_{i2} = h(\mathcal{H}(\hat{b}_i))$, where $\hat{b}_i$ — created and sent from the owner — is the new block to be inserted in copy

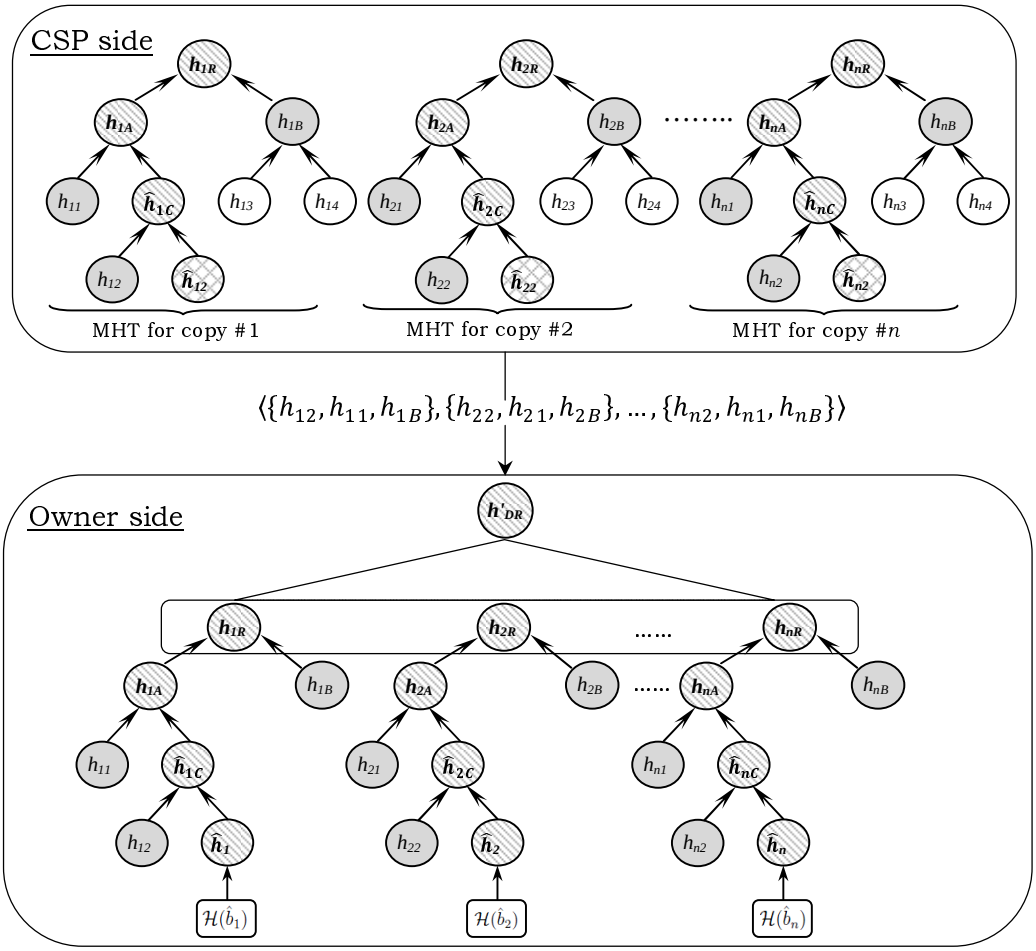


Fig. 14: Effect of block insertion operation on the MHTs and the directory root.

i: $1 \leq i \leq n$. The dashed nodes indicate the tree nodes that are updated due to the insertion of the new block. The updated hash values of these nodes are computed as h(left child||right child). The nodes $\{\hat{h}_{iC}\}_{1\leq i\leq n}$ are generated to re-arrange the structure of the MHTs according to the newly added leaf nodes. The grey nodes indicate the authentication paths of the newly inserted blocks, e.g., $\{h_{12}, h_{11}, h_{1B}\}$ is the authentication path of the new block $\hat{b}_1$. *On the owner side*, the owner uses the authentication paths $\langle\{h_{12}, h_{11}, h_{1B}\}, \{h_{22}, h_{21}, h_{2B}\}, \ldots, \{h_{n2}, h_{n1}, h_{nB}\}\rangle$ sent from the CSP and the new blocks $\{\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n\}$ to generate the new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$. The dashed nodes indicate the generated cryptographic hashes using the authentication paths (grey circles) sent from the CSP and the new blocks $\{\hat{b}_i\}_{1\leq i\leq n}$. Thus, $\hat{h}_1 = h(\mathcal{H}(\hat{b}_1)), \hat{h}_{1C} = h(h_{12}||\hat{h}_1), h_{1A} = h(h_{11}||\hat{h}_{1C})$, and $h_{1R} = h(h_{1A}||h_{1B})$. The computation of $\{h_{iR}\}_{2\leq i\leq n}$ are done the same way. The owner uses the computed $\{h_{iR}\}_{1\leq i\leq n}$ to generate the updated directory root h'_{DR}, and finally computes the updated metadata $\mathcal{M}' = h(ID_F||h'_{DR})$.

- **Deletion**. Fig. 15 shows that the second block is to be deleted from all copies outsourced to the CSP. *On the CSP side*, the leaf nodes with crosses indicate the nodes to be deleted. The fragmented curved arrows indicate that after deleting the specified leaf nodes, the nodes $\{h_{i1}\}_{1\leq i\leq n}$ replace the nodes $\{h_{iA}\}_{1\leq i\leq n}$, and thus the MHTs are re-arranged. The dashed nodes indicate the tree nodes that are updated due to the deletion of the second block. The updated hash values of these nodes are computed as h(left child||right child). The grey nodes indicate the authentication paths of the deleted blocks, e.g., $\{h_{11}, h_{1B}\}$ is the authentication path of the deleted block $\tilde{b}_{12}$. *On the owner side*, the owner uses the authentication paths $\langle\{h_{11}, h_{1B}\}, \{h_{21}, h_{2B}\}, \ldots, \{h_{n1}, h_{nB}\}\rangle$ sent from the CSP to generate the new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$. The dashed nodes indicate the generated cryptographic hashes using the authentication paths (grey circles) sent from

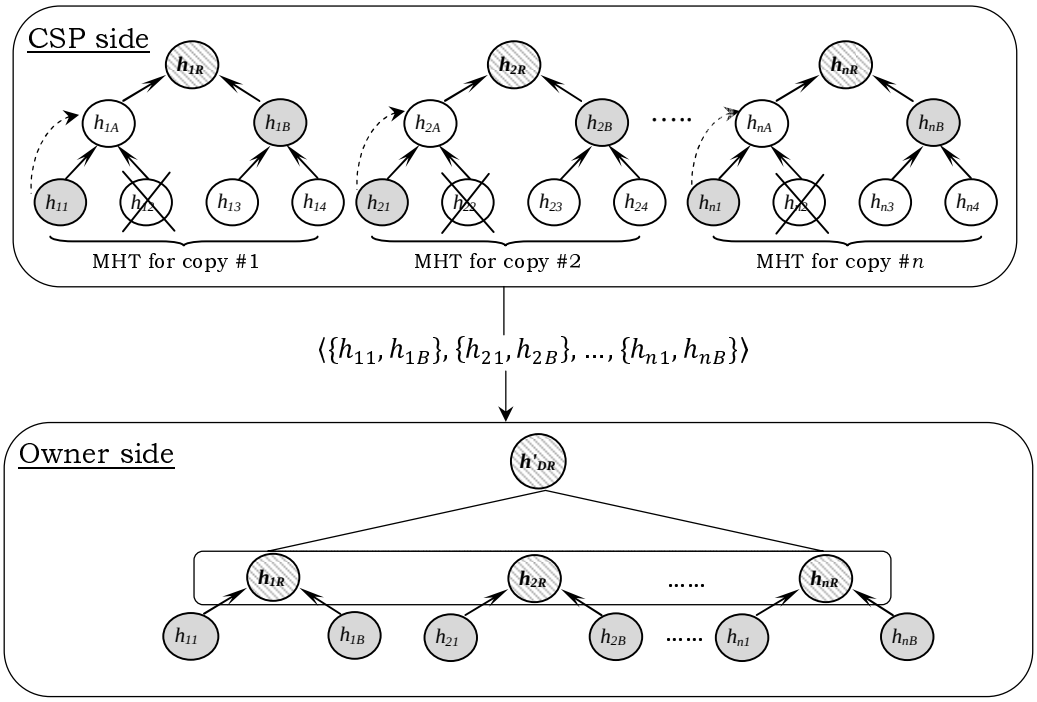


Fig. 15: Effect of block deletion operation on the MHTs and the directory root.

the CSP. Thus, $h_{iR} = h(h_{i1}||h_{iB}) : 1 \leq i \leq n$. The owner uses the computed $\{h_{iR}\}_{1\leq i\leq n}$ to generate the updated directory root h'_{DR}, and finally computes the updated metadata $\mathcal{M}' = h(ID_F||h'_{DR})$.

APPENDIX B
SECURITY ANALYSIS

In this section we present a formal security analysis for the proposed schemes. The essence of the security analysis for both schemes is similar, so we only present the security analysis for the MB-DMCPDP scheme. For the TB-DMCPDP scheme, we should note that the verification step of the metadata $\mathcal{M}$ will fail unless the CSP sends the correct information $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1\leq i\leq n \\ (j,*)\in Q}}$ along with the accurate authentication paths $\langle \mathbb{A}_{ij} \rangle_{\substack{1\leq i\leq n \\ (j,*)\in Q}}$ for the blocks being challenged. This is due to the collision resistance property of the cryptographic hash function h and the used sequence to reconstruct the directory root h_{DR}.

In our security analysis, we depend on the hardness of the computational Diffie-Hellman (CDH) problem and the discrete logarithm (DL) problem.

Definitions.

1) **CDH problem:** given $g,\ g^x,\ h \in \mathbb{G}$ for some group $\mathbb{G}$ and $x \in \mathbb{Z}_p$, compute h^x
2) **DL problem:** given $g,\ h \in \mathbb{G}$ for some group $\mathbb{G}$, find x such that $h = g^x$.

We will demonstrate that the cloud servers can provide valid responses to verifier's challenges only if they actually have all data copies in a consistent and uncorrupted state.

Theorem 1. *Assuming the hardness of both the CDH and the DL problems in bilinear groups, the verifier of the proposed MB-DMCPDP scheme accepts a response to a challenge vector only if a correctly computed proof* $\mathbb{P} = \{\sigma,\ \mu\}$, *where* $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n \\ 1\leq k\leq s}}$ *is sent from the CSP.*

Proof. First, since the ID_F is embedded into the block tag, the CSP cannot use blocks from different files and pass the auditing procedures even if the owner uses the same secret key x with all his files. Second, the goal of a malicious CSP is to generate a response that is not correctly computed and to pass the verification process.

We prove the theorem by contradiction. Let $\mathbb{P}' = \{\sigma',\ \mu'\}$ be the malicious CSP's response, where $\mu' = \{\mu'_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$. Let $\mathbb{P} = \{\sigma,\ \mu\}$ be the expected response from an honest CSP, where $\sigma = \prod_{(j,r_j)\in Q} \sigma_j^{r_j}$, $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$, and $\mu_{ik} = \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk}$. If the malicious CSP's response $\mathbb{P}' = \{\sigma',\ \mu'\}$ passes the verification equation (2), then we can find a solution to the CDH problem and the DL problem.

According to the correctness of MB-DMCPDP scheme, the expected proof $\mathbb{P} = \{\sigma,\ \mu\}$ satisfies the verification equation, i.e.,

$$\hat{e}(\sigma, g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu_{ik}}, y)$$

Assume that $\sigma' \neq \sigma$, and σ' passes the verification equation, then we have

$$\hat{e}(\sigma', g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu'_{ik}}, y)$$

Obviously, if $\mu'_{ik} = \mu_{ik}\ \forall(i,k)$, it follows from the above verification equations that $\sigma' = \sigma$ which contradicts our assumption. Let us define $\Delta\mu_{ik} = \mu'_{ik} - \mu_{ik}\ (1 \le i \le n, 1 \le k \le s)$. It must be the case that at least one of $\{\Delta\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$ is nonzero. Dividing the verification equation for the malicious response by the verification equation for the expected response, we obtain

$$\begin{aligned}
\hat{e}(\sigma' \cdot \sigma^{-1}, g) &= \hat{e}(\prod_{k=1}^{s} u_k^{\sum_{i=1}^n \Delta\mu_{ik}}, y)\\
\hat{e}(\sigma' \cdot \sigma^{-1}, g) &= \hat{e}(\prod_{k=1}^{s} u_k^{x\cdot\sum_{i=1}^n \Delta\mu_{ik}}, g)\\
\sigma' \cdot \sigma^{-1} &= \prod_{k=1}^{s} u_k^{x\cdot\sum_{i=1}^n \Delta\mu_{ik}}.
\end{aligned}$$

We set $u_k = g^{\alpha_k} h^{\beta_k}$ for $\alpha_k, \beta_k \in \mathbb{Z}_p$, and thus

$$\begin{aligned}
\sigma' \cdot \sigma^{-1} &= \prod_{k=1}^{s} (g^{\alpha_k} h^{\beta_k})^{x\cdot\sum_{i=1}^n \Delta\mu_{ik}}\\
\sigma' \cdot \sigma^{-1} &= \prod_{k=1}^{s} (y^{\alpha_k} h^{x.\beta_k})^{\sum_{i=1}^n \Delta\mu_{ik}}\\
\sigma' \cdot \sigma^{-1} &= y^{\sum_{k=1}^s \alpha_k\cdot\sum_{i=1}^n \Delta\mu_{ik}} \cdot h^{x\cdot\sum_{k=1}^s \beta_k\cdot\sum_{i=1}^n \Delta\mu_{ik}}\\
h^x &= (\sigma' \cdot \sigma^{-1} \cdot y^{-\sum_{k=1}^s \alpha_k\cdot\sum_{i=1}^n \Delta\mu_{ik}})^{\frac{1}{\sum_{k=1}^s \beta_k\cdot\sum_{i=1}^n \Delta\mu_{ik}}}.
\end{aligned}$$

Hence, we have found a solution to the CDH problem unless evaluating the exponent causes a division by zero. However, we noted that not all of $\{\Delta\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$ can be zero and the probability that $\beta_k = 0$ is $\frac{1}{p}$ which is negligible. Therefore, if $\sigma' \neq \sigma$, we can use the malicious CSP to break the CDH problem, and thus we guarantee that σ' must be equal to σ.

It is only the values $\mu' = \{\mu'_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$ and $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$ that can differ. Assume that the malicious CSP responds with $\sigma' = \sigma$ and $\mu' \neq \mu$. Now we have

$$\hat{e}(\sigma, g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu_{ik}}, y) =$$

$$\hat{e}(\sigma', g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu'_{ik}}, y)$$

from which we conclude that

$$\hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu_{ik}}, y) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{VN}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu'_{ik}}, y).$$

Thus,

$$\begin{aligned} 1 &= \hat{e}(\prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \Delta\mu_{ik}}, y) \\ 1 &= \prod_{k=1}^{s} (g^{\alpha_k} h^{\beta_k})^{\sum_{i=1}^{n} \Delta\mu_{ik}} \\ 1 &= g^{\sum_{k=1}^{s} \alpha_k \cdot \sum_{i=1}^{n} \Delta\mu_{ik}} \cdot h^{\sum_{k=1}^{s} \beta_k \cdot \sum_{i=1}^{n} \Delta\mu_{ik}} \\ h &= g^{-\frac{\sum_{k=1}^{s} \alpha_k \cdot \sum_{i=1}^{n} \Delta\mu_{ik}}{\sum_{k=1}^{s} \beta_k \cdot \sum_{i=1}^{n} \Delta\mu_{ik}}}. \end{aligned}$$

Now, we have found a solution to the DL problem unless evaluating the exponent causes a division by zero. However, the probability that $\beta_k = 0$ is $\frac{1}{p}$ which is negligible. Therefore, if there is at least one difference between $\{\mu'_{ik}\}_{\substack{1 \le i \le n \\ 1 \le k \le s}}$ and $\{\mu_{ik}\}_{\substack{1 \le i \le n \\ 1 \le k \le s}}$, we can use the malicious CSP to break the DL problem. As a result, we guarantee that $\{\mu'_{ik}\}$ must be equal to $\{\mu_{ik}\}$ $\forall(i,k)$. $\square$