

Rainbow Over Clouds: A Lightweight Pairing-free Multi-Replica Multi-Cloud Public Auditing Scheme

Reyhaneh Rabaninejad¹, Antonis Michalas^{1,2}, and Salil S. Kanhere³

¹ Tampere University, Tampere, Finland
reyhaneh.rabbaninejad@tuni.fi
antonios.michalas@tuni.fi

² University of Westminster, U.K.

³ University of New South Wales, Sydney, Australia
salil.kanhere@unsw.edu.au

Abstract. To enhance data availability and durability, data replication across multiple cloud servers has become a widely adopted strategy among data owners. The Proof of Replication (PoRep) technique is often used to ensure the integrity of multi-replicas stored in multi-cloud servers in distinct, dedicated storage, effectively preventing collusion where multi-cloud servers might store only a single copy of the file. Various PoRep schemes also known as multi-replica Provable Data Possession (PDP) have been proposed and applied to different scenarios. While these protocols are more computationally efficient than applying single-replica PDP schemes to each file copy individually, they still incur significant computational overhead in generating tags for all replicas and lack explicit mechanisms for fault localization. In this paper, we introduce Rainbow-PDP, a lightweight, provably secure multi-replica, multi-cloud public auditing scheme without pairing. Rainbow-PDP allows data owners to generate multiple replicas from an original file with minimal computational cost. Notably, it eliminates the need to generate verification tags on replicas distributed across various cloud providers. Instead, data integrity is assured by comparing proofs on different replicas during the verification phase. In cases of verification failure, the system can also locate corrupted replicas. Rainbow-PDP is proved to be secure against untrusted colluding servers. Experimental results demonstrate that Rainbow-PDP offers significant efficiency gains for both data owners and public verifiers when compared to existing state-of-the-art solutions. We also explore modifications to support efficient dynamic operations on data blocks, further enhancing Rainbow-PDP's practical applicability.

Keywords: Cloud Storage · Collusion Resistant · Data Outsourcing · Multi-Cloud Servers · Multi-Replica · Public Auditing · Proof of Replication

1 Introduction

Cloud storage, has attracted a lot of interest in recent years by offering efficient and scalable services with lower cost. Data owners outsource their data to cloud servers instead of keeping them in local systems and authorized users from different places can remotely access the outsourced data. However, loss of direct control over the data,

raises several data security and privacy challenges. Some untrusted cloud servers may put integrity of outsourced data at risk by manipulating the data or even deleting rarely accessed parts of the data to free up some storage space. The notion of Provable Data Possession (PDP) introduced by Ateniese *et al.* [2], guarantees the integrity of outsourced data. In PDP protocols, the data file is divided into several blocks, and the data blocks together with some metadata (tags) generated on them are outsourced to cloud servers. These tags help the verifier to audit data blocks in verification phase without downloading the data. In order to audit data integrity, verifier sends a small random subset of block indices as a challenge to the server who returns a proof on the specified data blocks that are hosted by it. Finally, verifier checks the returned proof thus confirming data integrity. Extensive research has been done in this context in recent years (e.g. [4, 6, 10–14, 17, 19, 21]). However, all of the above solutions only audit the integrity of a *single file copy* stored at a cloud server. To increase the availability and durability guarantees on the outsourced data, multi-replica PDP schemes were proposed by Curtmola *et al.* [5]. In a multi-replica PDP scheme, each data file is replicated and multiple replicas are stored on multiple cloud servers. In case a file copy is corrupted or even if a server storing some replicas is inaccessible, data is reconstructed using data replicas hosted on other servers. Multi-replica PDP schemes check the integrity of all file replicas stored on different cloud servers simultaneously and efficiently. A multi-replica PDP protocol should meet the following main requirements [7]: (1) Outsourced file replicas are unique and distinct to prevent collusion of untrusted servers. Otherwise, if file replicas are identical, colluding servers may free up some storage space and only store one replica altogether, without being detected. (2) The protocol verifies all file replicas by one challenge-response execution. Separate verification of file replicas makes the scheme inefficient. (3) The protocol allows public auditing so that anyone can audit the integrity of outsourced files on behalf of data owners. A naive solution for multi-replica auditing is to leverage a single-replica PDP scheme N times to separately audit N file replicas. MR-PDP scheme proposed in [5] is computationally more efficient compared with this trivial solution. In MR-PDP, the data owner first encrypts the original file and masks the encrypted file blocks with randomness generated by employing different Pseudo-Random Functions (PRF) to generate distinct file replicas. The data owner also generates a set of tags on encrypted file blocks and outsources them along with N file replicas to storage servers. All file replicas are audited through one challenge-response interaction and in case some replicas are lost/corrupted, MR-PDP can generate further replicas on demand with low cost. However, MR-PDP scheme does not provide the third property discussed above (public auditing) and only supports private verification. Various categories of Proofs of Replication (PoRep), such as private/public replication and private/public verification, have been explored in the literature. In private replication, the client generates the encoded replicas during a pre-processing phase, while public replication involves outsourcing the entire replication process to a server. These categories include private replication-private verification [1, 5], private replication-public verification [15], and public replication-public verification [9, 16]. Public replication is achieved by not using any secrets in file replication, making the process transparent as defined by Filecoin [9]. To counteract the generation attack, where a server generates a file replica

on-the-fly when creating a proof, Lerner [16] used time-asymmetric encodings to slow down the replication process. Filecoin [9] introduced “Seal”, slowable pseudo-random permutations constructed using a block cipher in cipher block chaining mode, made publicly verifiable with SNARKs [3]. This slow replication property makes it nearly impossible for an adversary to generate replicas just-in-time when responding to a challenge.

Slow encoding schemes rely on timing assumptions. PoRep schemes that use timing assumptions require users to set a security parameter for the verifier timeout and necessitate the prover to submit proofs within a specific time frame. However, an adversary with substantial computational power can still generate a correct proof before the timeout. To maintain security, users must set a large security parameter, which significantly increases computational costs when creating file replications. The high computational cost of generating tags for each file replica makes these schemes impractical for real-world applications. Additionally, most existing PoRep schemes do not explicitly address fault localization. In the event of audit faults, if the faulty replication cannot be quickly identified, all replications must be verified sequentially, resulting in a highly inefficient verification process. These challenges led us to explore a more practical solution for secure, lightweight multi-replica, multi-cloud storage auditing with public verifiability. Our approach does not rely on timing assumptions and ensures rapid fault localization.

1.1 Contributions

In this paper, we propose Rainbow-PDP: a provable secure lightweight multi-replica multi-cloud public auditing scheme without pairing⁴. The name is inspired by the different colors of rainbow depicting different file replicas. The main contributions of Rainbow-PDP can be summarized as follows:

- C1.** In Rainbow-PDP, the data owner generates different replicas from the original data file in a computationally efficient manner leveraging the proposed linear encoding algorithm. Moreover, no verification tags are generated on file replicas outsourced to different storage servers. Specifically, verification algorithm uses the idea of comparing proofs on distinct encoded replicas, instead of verifying all storage proofs separately, to ensure data integrity in the verification phase. Additionally, in case of verification failure, the verifier can quickly and efficiently locate the corrupted replicas.
- C2.** Rainbow is formally proved to achieve soundness in the presence of colluding malicious servers. From a performance perspective, experimental results showcase the scalability and efficiency of our scheme for both the data owner and the public verifier. The data owner and verification costs in Rainbow are independent of file size, which enhances its scalability with regards to large files. Furthermore, tag-free and pairing-free structure of Rainbow-PDP makes the protocol lightweight and suitable for entities with weak resources (e.g. commodity smartphones).

⁴ Pairing, also referred to as a Bilinear Map [8], is a fundamental operation in numerous cryptographic protocols, commonly known as pairing-based cryptographic schemes. However, it is associated with significant implementation overhead, making it computationally expensive.

1.2 Paper Organization

The rest of the paper is organized as follows. In [section 2](#), we discuss system and security model and notations used throughout the paper. In [section 3](#), a detailed design of RainBow-PDP is proposed. Security of our mechanism is proved in [section 4](#) while [section 5](#) gives the performance evaluation. [section 6](#) investigates dynamic RainBow-PDP, and finally [section 7](#) concludes the paper.

2 Problem Statement

This section introduces the system model, scheme outline, and the security model used in this paper.

2.1 System Model

The system model of our multi-replica PDP scheme as shown in [Figure 1](#) consists of four entities: Data Owner (DO), Cloud Orchestrator (CO), Cloud Storage Provider (CSP), and Third Party Auditor (TPA).

- DO, who wishes to outsource a data file to the cloud performs the following tasks: (1) Divides the data file into a number of blocks and encrypts each individual block using a secure symmetric cipher; (2) Generates several different replicas based on the encrypted original file; (3) Sends file replicas to CO and deletes all data from his local storage.
- CO serves as the intermediary between CSPs and DO/TPA. Upon receiving file replicas from the DO, CO distributes replicas to target CSPs. Also in the auditing phase, CO receives the challenge from TPA and forwards it to the target CSPs. CO next collects the returned storage proofs and sends them to the TPA.
- CSP possesses a pool of storage/computation resources to provide storage services to DOs and save their data. It also generates proof of data possession on the TPA's demand.
- TPA audits the integrity of all outsourced replicas on behalf of the DO. To this end, the TPA sends a challenge to CO who forwards it to the corresponding CSPs. The CSPs generate proofs on their stored replicas based on the challenged blocks and send them back to CO. The CO then forwards all collected proofs to the TPA. Finally, the TPA verifies the proofs and identifies the corrupted replicas in case the verification fails.

2.2 Scheme Outline

Building on the previously discussed system model, we now present the algorithmic structure and protocol syntax of a multi-replica PDP scheme. The Setup, KeyGen, and RepGen algorithms are carried out by the DO. The Challenge and ProofVerify algorithms are performed by the TPA, while the ProofGen algorithm is jointly executed by CSPs and CO.

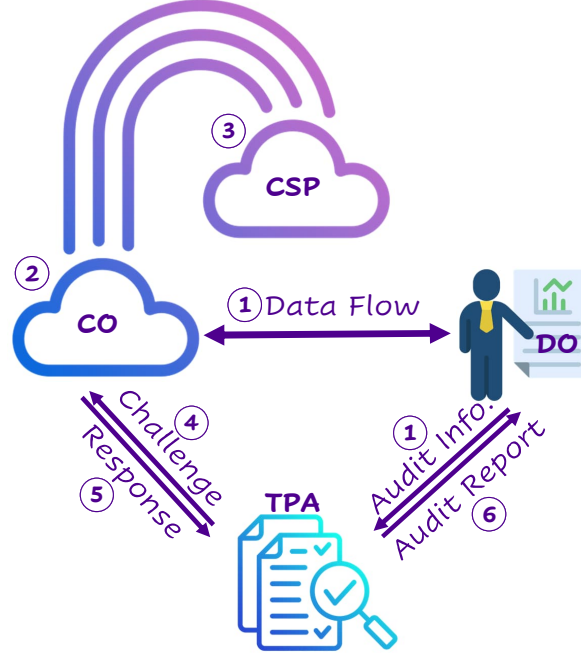


Fig. 1: The system model in this paper. ① Data Owner (DO) sends file replicas to Cloud Orchestrator (CO). ② CO distributes file replicas to available Cloud Service Providers (CSPs). ③ CSPs store the file replicas. ④ Third-Party Auditor (TPA) sends a challenge to CO. ⑤ CO responds with proofs collected from CSPs, which TPA verifies. ⑥ TPA sends the audit report to DO.

Definition 1 (Multi-Replica PDP).

- $\text{Setup}(1^k) \rightarrow \text{param}$. The algorithm input is security parameter 1^k and its output is the system parameters param .
- $\text{KeyGen}(1^k, \text{param}) \rightarrow (k_1, k_2, K)$. DO runs this algorithm with inputs security parameter 1^k and public parameters param , and generates private parameters prv .
- $\text{RepGen}(\text{prv}, F^*) \rightarrow \{F_i\}_{1 \leq i \leq N}$. DO executes this algorithm by taking private parameters prv and original data file F^* as input and generates N different file replicas $\{F_i\}_{1 \leq i \leq N}$ as output.
- $\text{Challenge}(\text{param}) \rightarrow \text{chal}$. The TPA runs this algorithm on input param and outputs a challenge chal to check all file replicas' integrity.
- $\text{ProofGen}(\text{param}, \text{chal}, \{F_i\}_{1 \leq i \leq N}) \rightarrow \{\pi_i\}_{1 \leq i \leq N}$. Each CSP executes this algorithm to generate proof for integrity of all replicas it stores. The inputs are file replica F_i , public parameters param , and challenge chal , and the output is π_i as the integrity proof for replica F_i . The CO collects all integrity proofs $\{\pi_i\}_{1 \leq i \leq N}$.

- **ProofVerify** ($\text{param}, \text{chal}, \{\pi_i\}_{1 \leq i \leq N}$) $\rightarrow \{0, 1\}$. The TPA who runs this algorithm takes param , chal , and $\{\pi_i\}_{1 \leq i \leq N}$. The verification passes if all proofs $\{\pi_i\}_{1 \leq i \leq N}$ are valid and it outputs 1; otherwise it outputs 0.

2.3 Security Model

In PDP mechanisms, the CO and CSPs are not considered fully trusted in the sense that they may try to hide data loss/corruption from the TPA in order to maintain their reputation. Besides, the TPA is assumed to be honest-but-curious meaning that, it honestly executes all the protocol steps and returns the real auditing report to the DO. However, it may try to learn information about the data content.

A PDP scheme should provide correctness and soundness as defined below.

Definition 2 (Correctness). *If CO, CSPs and the TPA honestly follow the prescribed protocol actions, then the proofs returned by the CO must pass the verification performed by the TPA.*

Soundness. This property guarantees that untrusted CO and CSPs cannot mislead the TPA into passing verification when the data has been compromised. Figure 2 outlines the detailed soundness game setup between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$.

Definition 3 (Soundness). *A Multi-Replica PDP (MR-PDP) scheme achieves soundness if for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$*

$$\text{Adv}_{MR-PDP, \mathcal{A}}^{\text{SND}}(k) := \Pr[\text{SND}_{MR-PDP, \mathcal{A}}(k) \rightarrow 1] \leq \text{negl}(k),$$

where $\text{SND}_{MR-PDP, \mathcal{A}}(k)$ is defined in Figure 2.

Setup Phase. Challenger $\mathcal{B}$ runs the Setup and KeyGen algorithms to generate private parameters prv and the public parameters param . He then forwards param to $\mathcal{A}$.

Query Phase. $\mathcal{A}$ can adaptively query $\mathcal{B}$ as follows: $\mathcal{A}$ sends a query on input (F^*, N) to obtain N replicas of original file F^* it selects (RepGen Query). The challenger generates N different file replicas $\{F_i\}_{1 \leq i \leq N}$ using RepGen algorithm and forwards them to $\mathcal{A}$. $\mathcal{A}$ stores file F^* and its corresponding replicas $\{F_i\}_{1 \leq i \leq N}$. At any time, $\mathcal{A}$ can choose to continue the query phase or go to the next phase.

Challenge Phase. Challenger $\mathcal{B}$ generates a chal by running Challenge algorithm to audit a file F^* which has been queried in the Query phase and sends chal to $\mathcal{A}$. $\mathcal{A}$ outputs a proof of possession under chal by executing ProofGen algorithm. The adversary is provided with the result of ProofVerify algorithm which checks the proof validity.

Forgery Phase. Finally, the adversary $\mathcal{A}$ outputs proof P on all replicas of some file in its storage. $\mathcal{A}$ wins the game if ProofVerify algorithm on P passes and P is not equal to local proof computed by challenger $\mathcal{B}$ using ProofGen algorithm on real data.

Fig. 2: $\text{SND}_{MR-PDP, \mathcal{A}}(k)$ experiment.

3 RainBow-PDP Construction

In this section, we present the construction of RainBow-PDP which consists the main contribution of this paper. RainBow-PDP is a tuple of six algorithms (Setup, KeyGen, RepGen, Challenge, ProofGen, ProofVerify). Table 1, provides a list of all notations used in RainBow-PDP.

Table 1: Notation

Notation	Definition
G	A multiplicative cyclic group
q	The prime order of group G
q'	A prime much smaller than q .
g	Generator of group G
k	Security parameter
ψ, θ	Two pseudo-random functions defined as $\psi : Z_q^* \times \{0, 1\}^{\log_2 N} \rightarrow Z_q^*$ and $\theta : Z_q^* \times \{0, 1\}^{\log_2 n + \log_2 N} \rightarrow Z_q^*$
E_K	A secure symmetric cipher
F^*	The original data file divided into n blocks $(b_1, \dots, b_n)$
F	The encrypted data file denoted as $F = (m_1, \dots, m_n)$
$\{F_i\}_{1 \leq i \leq N}$	The set of N file replicas, where the i^{th} replica is denoted as $F_i = (m_{i1}, \dots, m_{in})$
d	Number of servers storing file replicas
$\{L_1, \dots, L_d\}$	Set $L = [1, N]$ is divided into d subsets $\{L_1, \dots, L_d\}$, where L_k includes the replica indices stored at server S_k
N_k	Number of elements in subset L_k
ρ	Secret random element chosen by TPA in challenge phase

Setup : This algorithm takes the security parameter 1^k , and selects multiplicative cyclic group G with prime order $q = \Theta(2^k)$ and g as generator of G . We also define two PRFs $\psi : Z_q^* \times \{0, 1\}^{\log_2 N} \rightarrow Z_q^*$ and $\theta : Z_q^* \times \{0, 1\}^{\log_2 n + \log_2 N} \rightarrow Z_q^*$ and a secure symmetric cipher $E_K(\cdot)$. The public parameters are $\text{param} = (q, G, g, \psi, \theta, E_K)$. A data file F^* is divided into n blocks $b_i \in \mathbb{Z}_q$ denoted as $F^* = (b_1, \dots, b_n)$.

KeyGen : DO selects two random seeds $k_1, k_2 \in Z_q^*$ as the input secret keys to the pseudo-random functions ψ and θ , respectively. He also chooses K as the secret key of the symmetric encryption scheme $E_K(\cdot)$. DO shares k_1, k_2 with the TPA via a secure channel but keeps $\text{prv} = K$ confidential.

RepGen : DO first encrypts the original file F^* and obtains the encrypted file $F = (m_1, \dots, m_n)$, where m_j is symmetric encryption of block b_j , $m_j = E_K(b_j)$, for all

$1 \leq j \leq n$. Next, DO generates N distinct replicas of file F denoted as $\{F_i\}_{1 \leq i \leq N}$, where the i^{th} replica $F_i = (m_{i1}, \dots, m_{in})$ is generated as follows:

1. $\alpha_i = \psi_{k_1}(i)$
2. $\beta_{ij} = \theta_{k_2}(i \parallel j)$, for all $1 \leq j \leq n$
3. $m_{ij} = \alpha_i m_j + \beta_{ij} \pmod{q}$, for all $1 \leq j \leq n$.

Assume d as the number of CSPs which are going to store N file replicas. DO divides the set $L = [1, N]$ into d subsets $L = \{L_1, \dots, L_d\}$, where L_k includes the replica indices to be stored at server S_k . Assume the number of elements in subset L_k is N_k . Obviously, $N = \sum_{k=1}^d N_k$, $L = \cup_{k=1}^d L_k$, and for $i \neq j$: $L_i \cap L_j = \emptyset$. The DO sends $(\{F_i\}_{1 \leq i \leq N}, \{L_k\}_{1 \leq k \leq d})$ to CO who forwards file replicas which indices are the subset L_k to the corresponding server S_k . The DO may then delete the original file and the replicas from its local storage.

It should be noted that RepGen algorithm can be divided into two **online/offline** phases. That is, since the pseudo-random functions ψ, θ are independent of the file content, they can be calculated offline. Therefore, the online phase of RepGen only contains computing the linear combination $m_{ij} = \alpha_i m_j + \beta_{ij} \pmod{q}$ which has lightweight operations (modular addition/multiplication) and can be executed very fast.

Challenge : To challenge the integrity of all copies of file F , the TPA selects a random c -element subset $J \subset [1, n]$ as the block indices to be challenged in the auditing process and for each $j \in J$ chooses a random value $y_j \in \mathbb{Z}_q$. It also picks random $\rho \in \mathbb{Z}_q$ and sends $chal = (\{(j, y_j)\}_{j \in J}, C = g^\rho)$ to CO who sends it to all d servers storing at least one copy of file F .

ProofGen : Server S_k , based on the received challenge $chal = (\{(j, y_j)\}_{j \in J}, C = g^\rho)$, generates an integrity proof for each and every file replica F_i where $i \in L_k$, as follows:

1. Computes $\mu_i = \sum_{j \in J} y_j m_{ij} \pmod{q}$, as the combination of the challenged blocks.
2. Computes $\pi_i = C^{\mu_i}$ and sends back $\{\pi_i\}_{i \in L_k}$ as all proofs of replicas it stores to CO.

Finally, CO collects all proofs $\{\{\pi_i\}_{i \in L_k}\}_{1 \leq k \leq d} = \{\pi_i\}_{1 \leq i \leq N}$ and sends $\{\pi_i\}_{1 \leq i \leq N}$ back to the TPA.

ProofVerify : Upon receiving proof $\{\pi_i\}_{1 \leq i \leq N}$, the TPA first computes $\alpha_i = \psi_{k_1}(i)$, $\beta_{ij} = \theta_{k_2}(i \parallel j)$, for all $1 \leq i \leq N$ and $1 \leq j \leq n$. This step is a one-time computation for all challenges. Next, it verifies the proof through the following procedure.

1. For all $1 \leq i \leq N$ computes: $\beta_i = \sum_{j \in J} y_j \beta_{ij} \pmod{q}$ and $Z_i = (\frac{\pi_i}{C^{\beta_i}})^{\frac{1}{\alpha_i}}$
2. The verification passes if $Z_1 = Z_2 = \dots = Z_N$, and fails otherwise.

RainBow-PDP also locates the corrupted replicas in case the ProofVerify algorithm fails. Specifically, if verification fails and the value Z_l is not equal to other values, it can be deduced that the corresponding replica l is corrupted.

Remark. We highlight that there is no TagGen algorithm in RainBow-PDP and no verification tags are generated on the file blocks in our scheme. Technically, the designed comparison between proofs returned on different replicas acts instead of tags to ensure data integrity in RainBow-PDP. As a result, the computation/communication overheads of generating/sending tags which is a great cost in previous schemes is omitted here. More details will be presented in the experimental results section (section 5).

4 Security Analysis of RainBow-PDP

In this section, we show correctness and prove security against server (soundness) for RainBow-PDP. We also discuss security of the protocol against colluding servers.

Theorem 1. [Correctness] *If the CO, the CSPs and the TPA honestly follow the protocol, the generated proof can pass ProofVerify in RainBow-PDP as in Definition 2.*

Proof. To show this, without loss of generality we show two values Z_i, Z_k corresponding to two proofs π_i, π_k generated based on protocol are equal:

$$\begin{aligned}
 Z_i &= \left(\frac{\pi_i}{C^{\beta_i}} \right)^{\frac{1}{\alpha_i}} = \left(\frac{C^{\mu_i}}{C^{\beta_i}} \right)^{\frac{1}{\alpha_i}} \\
 &= \left(\frac{C^{\sum_{j \in J} y_j m_{ij}}}{C^{\beta_i}} \right)^{\frac{1}{\alpha_i}} \\
 &= \left(\frac{C^{\sum_{j \in J} y_j (\alpha_i m_j + \beta_{ij})}}{C^{\beta_i}} \right)^{\frac{1}{\alpha_i}} \\
 &= \left(\frac{C^{\alpha_i \sum_{j \in J} y_j m_j + \sum_{j \in J} y_j \beta_{ij}}}{C^{\beta_i}} \right)^{\frac{1}{\alpha_i}} \\
 &= \left(\frac{C^{\alpha_i \sum_{j \in J} y_j m_j + \beta_i}}{C^{\beta_i}} \right)^{\frac{1}{\alpha_i}} \\
 &= \left(C^{\alpha_i \sum_{j \in J} y_j m_j} \right)^{\frac{1}{\alpha_i}} \\
 &= C^{\sum_{j \in J} y_j m_j}
 \end{aligned} \tag{1}$$

Writing the same equalities for Z_k also yields $C^{\sum_{j \in J} y_j m_j}$, which shows that if the proofs are generated based on intact file replicas, then the values Z_i and Z_k would be equal.

Theorem 2. [Soundness] *If ψ and θ are secure PRFs, SE is IND-CPA secure, Rainbow satisfies soundness as in Definition 3 in the random oracle model.*

Proof. The public verifier sends a challenge $(\{(j, y_j)\}_{j \in J}, C)$ to CO who forwards it to all servers. A valid response to the challenge for replica $F_i = (m_{i1}, \dots, m_{in})$ has the form $\pi_i = C^{\mu_i}$, where $\mu_i = \sum_{j \in J} y_j m_{ij}$. But, here we assume that the CSP storing replica F_i cheats and sends a forged proof $\pi'_i = C^{\mu'_i}$, where $\mu' = \sum_{j \in J} y_j m'_{ij} = \sum_{j \in J} y_j (m_{ij} + \Delta_j)$. The vector $\Delta = (\Delta_1, \Delta_2, \dots, \Delta_n)$ is nonzero which means at least one block m'_{ij} is modified. The Z values (which were introduced in ProofVerify algorithm) for the valid proof $\pi_i = C^{\mu_i}$ and the forged proof $\pi'_i = C^{\mu'_i}$ are as follows:

$$Z_i = \left(\frac{\pi_i}{C^{\beta_i}} \right)^{\frac{1}{\alpha_i}}, Z'_i = \left(\frac{\pi'_i}{C^{\beta_i}} \right)^{\frac{1}{\alpha_i}} \tag{2}$$

Now, since both proofs π_i, π'_i pass the verification, we have $Z_i = Z'_i$ and so, the RHS of both equations are equal:

$$\begin{aligned}
\left(\frac{\pi_i}{C^{\beta_i}}\right)^{\frac{1}{\alpha_i}} &= \left(\frac{\pi'_i}{C^{\beta_i}}\right)^{\frac{1}{\alpha_i}} \\
\left(\frac{C^{\sum_{j \in J} y_j m_{ij}}}{C^{\beta_i}}\right)^{\frac{1}{\alpha_i}} &= \left(\frac{C^{\sum_{j \in J} y_j m'_{ij}}}{C^{\beta_i}}\right)^{\frac{1}{\alpha_i}} \\
\left(\frac{C^{\sum_{j \in J} y_j m_{ij}}}{C^{\beta_i}}\right)^{\frac{1}{\alpha_i}} &= \left(\frac{C^{\sum_{j \in J} y_j (m_{ij} + \Delta_j)}}{C^{\beta_i}}\right)^{\frac{1}{\alpha_i}} \\
\Rightarrow (C^{\sum_{j \in J} y_j \Delta_j})^{\frac{1}{\alpha_i}} &= 1 \\
\Rightarrow \sum_{j \in J} y_j \Delta_j &= 0
\end{aligned} \tag{3}$$

Without loss of generality, we assume there are $\varsigma \leq c$ nonzero elements in vector $\Delta = \{\Delta_j\}_{j \in J}$. Accordingly, number of tuples $(y_1, \dots, y_c)$ satisfying $\sum_{j \in J} y_j \Delta_j = 0$, where $y_j \in \mathbb{Z}_q$, is at most $q^{\varsigma-1}$. As $\{y_j\}_{j \in J}$ values are randomly selected by the TPA in each challenge, the probability of holding Equation 3 is at most $\frac{q^{\varsigma-1}}{q^c} \leq \frac{q^{c-1}}{q^c} = \frac{1}{q}$. Since q is a prime number with order $q = \Theta(2^k)$, $\frac{1}{q}$ is negligible and the proof completes.

Proof Sketch of Collusion Resistance. In Theorem 2 we proved that malicious CSPs have negligible advantage in forging a proof on corrupted data, and passing ProofVerify algorithm. However, consider a scenario where malicious CSPs collude and make same changes in all file replicas. Since the terms α_i differ for difference replicas $1 \leq i \leq N$, even if the same changes are made in replicas, Z_i values would not be equal. Specifically, lets assume a value δ is added to t^{th} block denoted as m_{it} , for all replicas $1 \leq i \leq N$. The modified block would be equal $m_{it} + \delta = (\alpha_i m_t + \beta_{it}) + \delta$ (since the servers do not know values α_i, β_{it} , they cannot modify the original block to $m_t + \delta$).

Now, in Equation 1 the sixth equality modifies to $(C^{\alpha_i \sum_{j \in J} y_j m_j + y_t \delta})^{\frac{1}{\alpha_i}} = C^{\sum_{j \in J} y_j m_j + \frac{y_t \delta}{\alpha_i}}$. Since α_i values are different for all replicas $1 \leq i \leq N$ and unknown to servers, the terms $Z_i = C^{\sum_{j \in J} y_j m_j + \frac{y_t \delta}{\alpha_i}}$ are not equal for different replicas and ProofVerify algorithm fails. Moreover, since each CSP stores a distinct replica, collusion among CSPs to share only one replica and claim all copies are stored correctly is ineffective due to proof of Theorem 2. Therefore, RainBow-PDP is resistant to collusion attacks.

5 Performance Evaluation

In this section, we first evaluate all the overheads in RainBow-PDP. Next, the experimental results corresponding to these overheads are proposed.

5.1 Overhead Analysis

Let *Mul*, *Exp* and *Pair* denote modular multiplications, exponentiations and pairings, respectively. Also, index of the operation represents group that the operation is

Table 2: Performance comparison

		MR-PDP [5]	ID-PMDP [7]	RainBow-PDP
DO Computation	RepGen	$E_K + Nn\mathcal{R}$	NE_K	$E_K + N(n+1)\mathcal{R}$
	TagGen	$2nExp_{Z_N^*} + nMul_Z + nH$	$2Nn(Exp_G + Mul_G)$	0
Server Computation		$(c-1).Mul_{Z_N^*} + Exp_{Z_N^*}$	$c.(Exp_G + Mul_G) + d.Mul_G$	$c.Mul_{Z_q} + Exp_G$
TPA Computation		$(2N + c + 1).Exp_{Z_N^*} + (cN + c + N).Mul_{Z_N^*} + c.H$	$(cN + s + 3).Exp_G + (cN + s).Mul_G + 3Pair$	$cN.Mul_{Z_q} + 2N.Exp_G + N.Mul_G$
Communication Cost		$\log_2 c + 2 q + 1024N$	$\log_2 c + 2 q + (2s + 2) q $	$c(n + q) + (N + 1) q $
Server Storage Cost		$N F + 1024n$	$N F + Nn q $	$N F $

–DO is for data owner. Parameters N , n , d , and c respectively denote number of: file replicas, blocks in each replica, servers storing file replicas and number of challenged blocks in an auditing task.

defined in, for example Mul_G means multiplication over G . $\mathcal{R}$ denotes random number generation operation which is implemented using pseudo-random functions. H denotes a secure deterministic hash-and-encode function that maps strings uniformly to QR_N . Furthermore, additions in $\mathbb{Z}_q$ are omitted from overhead analysis, since they have nearly negligible cost.

Computation Overhead. The computation for the data owner includes generating file replicas (RepGen) and generating tags on file blocks (TagGen). For RepGen, the data owner first encrypts each block of the original file. Next, generates N distinct copies of encrypted file F denoted as $\{F_i\}_{1 \leq i \leq N}$, where for each copy $\{F_i\}$, a random value $\alpha_i = \psi_{k_1}(i)$ and n random values $\beta_{ij} = \theta_{k_2}(i \parallel j), 1 \leq j \leq n$ are generated. Therefore, RepGen cost for data owner equals to $E_K + N(n+1)\mathcal{R}$. Besides, TagGen cost for data owner is 0 in RainBow-PDP, since no verification tags are generated on file blocks in our scheme.

The TPA first generates a set of c random values to challenge data integrity which introduces only a negligible cost. Second, it runs ProofVerify algorithm to check the proof returned by server, by computing $\beta_i = \sum_{j \in J} y_j \beta_{ij}$ and $Z_i = (\frac{\pi_i}{C^{\beta_i}})^{\frac{1}{\alpha_i}}$ for all $1 \leq i \leq N$ which has the total cost $cN.Mul_{Z_q} + 2N.Exp_G + N.Mul_G$.

The cost for generating each proof $\pi_i = C^{\mu_i}$ in ProofGen algorithm run by server is equal to $c.Mul_{Z_q} + Exp_G$. It should be noted that Table 2 represents the cost of generating a single proof for all three schemes.

Communication Overhead. The communication cost is introduced by challenge-response phase in the protocol. In challenge $chal = (\{(j, y_j)\}_{j \in J}, C = g^{\rho})$, the terms y_j, C are elements of $\mathbb{Z}_q$ and G . Hence, the communication cost is $c(|n| + |q|) + |q|$ bits, in which $|q|$ is the element length of $\mathbb{Z}_q$, and $|n|$ is the length of an index j . Also, proofs $\{\pi_i\}_{1 \leq i \leq N}$ have communication overhead equal to $N|q|$ bits.

Storage Overhead. The storage space required for storing N file copies is equal to $N|F|$. However, since in our scheme no verification tags are generated on the file blocks, nothing more than file replicas are stored at the cloud servers. So, the total storage overhead of RainBow-PDP is $N|F|$.

Table 2 summarizes the total overheads of RainBow-PDP discussed above and also makes a comparison with two well-known multi-replica PDP schemes.

5.2 Experimental Results

In this section, we evaluate the efficiency of our scheme which we implemented on a laptop with configuration Intel I7-3470 3.20 GHz processor, 4 GB memory and Windows 7 operating system using MIRACL library [18]. The computational costs for data owner, cloud storage providers and TPA are evaluated by performing experiments in this setting. We note that $E_K(\cdot)$ was implemented with AES-128 as a standard symmetric cipher. Furthermore, since in practice block ciphers are used in most instances where a pseudo-random function is needed, we also implemented pseudo-random functions with AES-128 in our experiments.

Figure 3 depicts the total costs (RepGen+TagGen) for processing the entire file F at data owner side by increasing the file size from 2 MB to 20 MB, with an increment of 2 MB in each test. Equivalently, number of file blocks n changes from 125,000 to 1,250,000, with an increment of 125,000 in each test. We also generated 20 file replicas in each experiment. Since the schemes have different orders we show them in two distinct plots for better visibility. As can be seen in Figure 3, RainBow-PDP has great advantage at user side compared with MR-PDP [5] and ID-PMDP [7]. For example, processing a file of size 20 MB requires 0.1787 seconds in RainBow-PDP, while this time is 9443.9 and 188100 seconds for MR-PDP and ID-PMDP, respectively. This confirms the lightweight behavior of RainBow-PDP which makes it suitable for resource-constrained users.

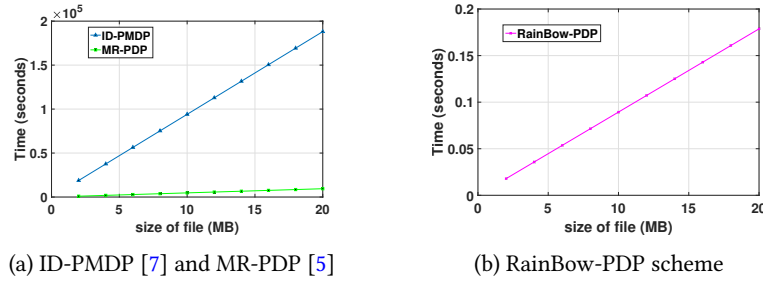


Fig. 3: Data owner processing time for the entire file

Figure 4 and Figure 5 illustrate experimental results at server and TPA sides, respectively. According to [2], if t fraction of the data file is corrupted, the probability of detecting corruption is $p = 1 - (1 - t)^c$, where c is the number of challenged

blocks in Challenge algorithm. For example, if 1% of a file is corrupted, for detection probability 0.9, $c = 230$ blocks should be challenged in the auditing process. Figure 4 demonstrates the simulation results of generating one proof in ProofGen algorithm at each server for detection probability changing from 0.5 to 0.99. We also considered 20 servers each storing one file replica (20 total replicas) in our simulations. From Figure 4, we see that RainBow-PDP and MR-PDP have similar proof generation times, and both are about 100 times more efficient than ID-PMDP for detection probability 0.9.

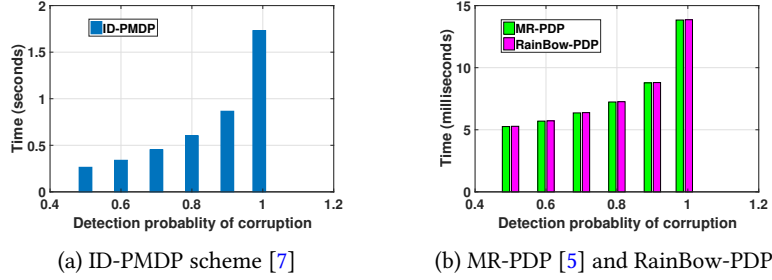


Fig. 4: Proof generation time at the server side for a 1% corrupted file

In Figure 5, auditing time at the TPA side for the tree protocols is compared. As can be seen in this figure, our scheme outperforms the two other schemes for computation cost at TPA side. Specifically, for detection probability 0.9, the TPA in RainBow-PDP requires 0.2512 seconds to simultaneously audit 20 file replicas, while the relevant times for MR-PDP and ID-PMDP are 1.1324 and 17.3547 seconds, respectively.

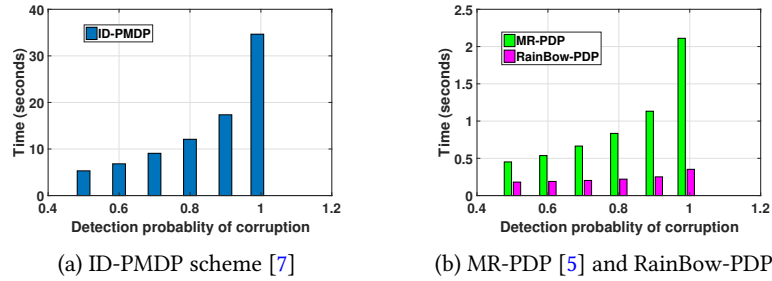


Fig. 5: Auditing time at the TPA side for a 1% corrupted file

6 Dynamic RainBow-PDP

In RainBow-PDP, the coefficients $\beta_{ij} = \theta_{k_2}(i \parallel j)$ depend on the index j of block m_{ij} . Therefore, RainBow-PDP does not support efficient dynamic operations on data

blocks. More precisely, when a user inserts or deletes a single block in outsourced file, all block indices after the inserted/deleted block change, and so their β_{ij} coefficients require re-computation which makes insert/delete operations expensive.

Here, we slightly modify RainBow-PDP to enable data owners (DOs) dynamically modify outsourced file blocks. This includes insert, delete or update operations on file blocks. To this end, we employ the concept of virtual index introduced in [20], [22]. Specifically, the coefficients are modified to $\beta_{ij} = \theta_{k_2}(i \parallel id_{ij})$, where $id_{ij} = \{v_{ij}, h_{ij}\}$ is named as identifier of block m_{ij} . Parameter v_{ij} is virtual index of block m_{ij} and $h_{ij} = H(m_{ij} \parallel v_{ij})$, where $H : \{0, 1\}^* \rightarrow \mathbb{Z}_{q'}$ is a secure hash function and q' is a prime much smaller than q . Virtual index of block m_{ij} is firstly set to $v_{ij} = j \cdot \delta$, where δ is a system parameter. Virtual indices determine the block orders and parameters h_{ij} make the block identifier id_{ij} dependent on the block content m_{ij} which makes the scheme secure against replace/replay attacks. More Technically, the details of Modify algorithm in Dynamic RainBow-PDP is as follows.

Modify.

–Insert.

1. To insert a block m'_{ij} into the outsourced file, DO first computes $v'_{ij} = (v_{ij-1} + v_{ij})/2$, $h'_{ij} = H(m'_{ij} \parallel v'_{ij})$ and $id'_{ij} = \{v'_{ij}, h'_{ij}\}$. Next, he sends m'_{ij} to the server and id'_{ij} to the TPA.
2. The server and the TPA respectively insert m'_{ij} and id'_{ij} in the j^{th} position related to file F_i . However, virtual indices and therefore identifiers of the blocks after the modified block remain unchanged. Therefore, β_{ij} coefficients do not require an update, and the insert operation is accomplished efficiently.

–Delete.

1. In order to delete block m_{ij} from the data stored at the server, DO sends the block index to the server and TPA.
2. The CSP and the TPA delete the corresponding position from their database. However, virtual indices of the blocks after the deleted block remain unchanged.

–Update.

1. To update the j^{th} block with a new value m'_{ij} , DO calculates $h'_{ij} = H(m'_{ij} \parallel v_{ij})$ and $id'_{ij} = \{v_{ij}, h'_{ij}\}$, with the same virtual index v_{ij} . Next, he sends m'_{ij} to the server and id'_{ij} to the TPA.
2. The server and the TPA update the j^{th} position related to file F_i with the new values m'_{ij} and id'_{ij} , respectively.

We note that the DO should make *same* modifications in all file replicas stored at different CSPs. If he fails to do so, the file replicas will not pass ProofVerify algorithm. Furthermore, in Dynamic RainBow-PDP, the TPA keeps a list of block identifiers $\{\{id_{ij}\}_{1 \leq j \leq n}\}_{1 \leq i \leq N}$. Later in ProofVerify algorithm, the TPA computes $\beta_{ij} = \theta_{k_2}(i \parallel id_{ij})$ and $\beta_i = \sum_{j \in J} y_j \beta_{ij}$ based on id_{ij} values in its storage. Therefore, Dynamic RainBow-PDP incurs an extra storage cost on the TPA side in comparison to the basic RainBow-PDP protocol. Besides, since the core algorithms of Dynamic RainBow-PDP (RepGen, ProofGen, ProofVerify) are the same as the ones in basic RainBow-PDP,

correctness and soundness of Dynamic RainBow-PDP can be shown through the same arguments as in Section 4 and hence we avoid repeating them here.

Remark. Table 3 makes a high-level comparison between RainBow-PDP and two selected prior protocols and gives an overview of important features that RainBow-PDP achieves.

Table 3: Feature list and comparison between RainBow-PDP and two existing well-known schemes

Feature \ Scheme	MR-PDP [5]	ID-PMDP [7]	RainBow-PDP
Data Dynamics	No	No	Yes
Identify Corrupted Replica	Yes	No	Yes
Lightweight User Computations	No	No	Yes ¹

¹this is because RainBow-PDP has no TagGen/Pairings in its design.

7 Conclusion

In this paper, we introduced Rainbow-PDP, a novel multi-replica public auditing scheme without pairing, designed to minimize computational overhead, particularly for data owners. The scheme enables data owners to generate multiple replicas of the original data file with minimal cost, without the need to create verification tags for file blocks. The tag-free and pairing-free structure of Rainbow-PDP makes it particularly well-suited for resource-constrained cloud users, such as those utilizing standard smartphones or other low-power devices. File replicas are outsourced to various storage providers, and during the verification phase, the auditor compares integrity proofs across the replicas, serving as a substitute for traditional tags to ensure data integrity. In the event of verification failure, the verifier can easily locate corrupted replicas. We provided a security proof demonstrating the scheme’s soundness. Performance evaluation confirm the superior efficiency of Rainbow-PDP compared to state-of-the-art methods. Additionally, we discussed how to support efficient dynamic operations on data blocks by making slight modifications to Rainbow-PDP.

References

1. Armknecht, F., Barman, L., Bohli, J.M., Karame, G.O.: Mirror: Enabling proofs of data replication and retrievability in the cloud. In: 25th USENIX Security Symposium (USENIX Security 16) (2016)
2. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM conference on Computer and communications security, ACM 2007. pp. 598–609. ACM (2007)

3. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for c: Verifying program executions succinctly and in zero knowledge. In: Annual cryptology conference. pp. 90–108. Springer (2013)
4. Cecchetti, E., Fisch, B., Miers, I., Juels, A.: Pies: Public incompressible encodings for decentralized storage. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1351–1367 (2019)
5. Curtmola, R., Khan, O., Burns, R., Ateniese, G.: Mr-pdp: Multiple-replica provable data possession. In: 2008 the 28th international conference on distributed computing systems. pp. 411–420. IEEE (2008)
6. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Annual Cryptology Conference. pp. 585–605. Springer (2015)
7. Li, J., Yan, H., Zhang, Y.: Efficient identity-based provable multi-copy data possession in multi-cloud storage. *IEEE Transactions on Cloud Computing* (2019)
8. Menezes, A.: An introduction to pairing-based cryptography. *Recent trends in cryptography* **477**, 47–65 (2009)
9. Protocol Labs: Filecoin: A decentralized storage network. <https://filecoin.io/filecoin.pdf> (2018)
10. Rabaninejad, R., Abdolmaleki, B., Malavolta, G., Michalas, A., Nabizadeh, A.: storna: Stateless transparent proofs of storage-time. In: European Symposium on Research in Computer Security. pp. 389–410. Springer (2023)
11. Rabaninejad, R., Asaar, M.R., Attari, M.A., Aref, M.R.: An identity-based online/offline secure cloud storage auditing scheme. *Cluster Computing* pp. 1–14 (doi:101007/s10586-019-03000-5, 2019)
12. Rabaninejad, R., Attari, M.A., Asaar, M.R., Aref, M.R.: A lightweight auditing service for shared data with secure user revocation in cloud storage. *IEEE Transactions on Services Computing* **15**(1), 1–15 (2019)
13. Rabaninejad, R., Attari, M.A., Asaar, M.R., Aref, M.R.: A lightweight identity-based provable data possession supporting users’ identity privacy and traceability. *Journal of Information Security and Applications* **51**, 102454 (2020)
14. Rabaninejad, R., Attari, M.A., Asaar, M.R., Aref, M.R.: Comments on a lightweight cloud auditing scheme: Security analysis and improvement. *Journal of Network and Computer Applications* **139**, 49–56 (doi:101016/j.jnca.201904012, 2019)
15. Rabaninejad, R., Liu, B., Michalas, A.: Port: non-interactive continuous availability proof of replicated storage. In: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. pp. 270–279 (2023)
16. Sergio Demian Lerner: Proof of unique blockchain storage. <https://bitslog.wordpress.com/2014/11/03/proof-of-local-blockchain-storage/> (2014)
17. Shacham, H., Waters, B.: Compact proofs of retrievability. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 90–107. Springer (2008)
18. Shamus Software Ltd: Miracl library. <http://www.shamus.ie/index.php?page=home>
19. Susilo, W., Li, Y., Guo, F., Lai, J., Wu, G.: Public cloud data auditing revisited: Removing the tradeoff between proof size and storage cost. In: European Symposium on Research in Computer Security. pp. 65–85. Springer (2022)
20. Wang, B., Li, B., Li, H.: Oruta: privacy-preserving public auditing for shared data in the cloud. *IEEE transactions on cloud computing* **2**(1), 43–56 (2014)
21. Yuan, H., Chen, X., Wang, J., Yuan, J., Yan, H., Susilo, W.: Blockchain-based public auditing and secure deduplication with fair arbitration. *Information Sciences* **541**, 409–425 (2020)
22. Zhu, Y., Wang, H., Hu, Z., Ahn, G.J., Hu, H., Yau, S.S.: Dynamic audit services for integrity verification of outsourced storages in clouds. In: Proceedings of the 2011 ACM Symposium on Applied Computing. pp. 1550–1557. ACM (2011)