Distrib Parallel Databases (2014) 32:507–534
DOI 10.1007/s10619-014-7151-0

Efficient integrity verification of replicated data in cloud using homomorphic encryption

Raghul Mukundan · Sanjay Madria · Mark Linderman

Published online: 24 June 2014

Abstract The cloud computing is an emerging model in which computing infrastructure resources are provided as a service over the internet. Data owners can outsource their data by remotely storing them in the cloud and enjoy on-demand high quality services from a shared pool of configurable computing resources. However, since data owners and the cloud servers are not in the same trusted domain, the outsourced data may be at risk as the cloud server may no longer be fully trusted. Therefore, data confidentiality, availability and integrity is of critical importance in such a scenario. The data owner encrypts data before storing it on the cloud to ensure data confidentiality. Cloud should let the owners or a trusted third party to check for the integrity of their data storage without demanding a local copy of the data. Owners often replicate their data on the cloud servers across multiple data centers to provide a higher level of scalability, availability, and durability. When the data owners ask the cloud service provider (CSP) to replicate data, they are charged a higher storage fee by the CSP. Therefore, the data owners need to be strongly convinced that the CSP is storing data copies agreed on in the service level contract, and data-updates have been correctly executed

Communicated by Dr. Shrideep Pallickara, Dr. Samee U. Khan, Dr. Rodrigo Calheiros, Dr. Rajkumar Buyya, Dr. Rui Zhang.

A preliminary version [16] of this paper has appeared in IEEE Data Engineering Bulletin, 2012.

R. Mukundan · S. Madria (✉)
Department of Computer Science, Missouri University of Science and Technology, Rolla, MO, USA
e-mail: madrias@mst.edu

R. Mukundan
e-mail: rmgq8@mst.edu

M. Linderman
Air Force Research Lab, Rome, NY, USA
e-mail: mark.linderman@rl.af.mil

on all the remotely stored copies. To deal with such problems, previous multi copy verification schemes either focused on static files or incurred huge update costs in a dynamic file scenario. In this paper, we propose a dynamic multi-replica provable data possession scheme (DMR-PDP) that while maintaining data confidentiality prevents the CSP from cheating, by maintaining fewer copies than paid for and/or tampering data. In addition, we also extend the scheme to support a basic file versioning system where only the difference between the original file and the updated file is propagated rather than the propagation of operations for privacy reasons. DMR-PDP also supports efficient dynamic operations like block modification, insertion and deletion on replicas over the cloud servers. Through security analysis and experimental results, we demonstrate that the proposed scheme is secure and performs better than some other related ideas published recently.

Keywords Replication · Data integrity · Cloud computing

1 Introduction

When users store data in the cloud, their main concern is about the security of the data, ability of the cloud to maintain data integrity and recovery of data in case of loss of data or sever failure. Cloud service providers (CSPs), in order to save storage cost, may tend to discard some data or data copies that are not accessed often, or mitigate such data to the second-level storage devices. CSPs may also conceal data loss due to management faults, hardware failures or attacks. Therefore, a critical issue in storing data at untrusted CSPs is periodically verifying whether the storage servers maintain data integrity and store data completely and correctly as stated in the service level agreement (SLA).

Replication is a commonly used technique to increase the data availability in the cloud computing. Cloud replicates the data and stores them strategically on multiple servers located at various geographic locations. Since the replicated data are copies, it is difficult to verify whether the cloud really stores multiple copies of the data. The cloud can easily cheat the owner by storing only one copy of the data. Thus, the owner would like to verify at regular intervals whether the cloud indeed possesses multiple copies of the data as claimed in the SLA. In general, the cloud has the capability to generate multiple replicas when a data owner challenges the CSP to prove that it possesses multiple copies of the data. Also, it is a valid assumption that the owner of the data may not have a copy of the data stored locally. So, the major task of the owner is not only to verify that the data is intact but also to recover the data if any deletions/corruptions of data are identified. If the owner, during his verification using DMR-PDP scheme, detects some data loss in any of the replicas in the cloud, he can recover the data from other replicas that are stored intact. Since, the replicas are to be stored at diverse geographic locations, it is assumed to be safe that the data loss will not occur at all the replicas at the same time.

Provable data possession (PDP) [1] is a technique to audit and validate the integrity of data stored on remote servers. In a typical PDP model, the data owner generates metadata/tag for a data file to be used later for integrity verification. To ensure security, the data owner encrypts the file and generates tags on the encrypted file. The data owner sends the encrypted file and the tags to the cloud, and deletes the local copy of the file. When the data owner wishes to verify data integrity, he generates a challenge vector and sends it to the cloud. The cloud replies by computing a response on the data and sends it to the verifier/data owner to prove that multiple copies of the data file are stored in the cloud. Different variations of PDP schemes such as [1,11,13,14,17,18,20,21,24] were proposed under different cryptographic assumptions. However, most of these schemes deal only with static data files and are valid only for verifying a single copy. A few other schemes such as [2,12,15,22,23] provide dynamic scalability of a single copy of a data file for various applications which means that the remotely stored data can not only be accessed by the authorized users, but also be updated and scaled by the data owner.

In this paper, we propose a scheme that allows the data owner to keep the data secure in the cloud, ensure data availability by creating multiple replicas and securely ensure that the CSP stores multiple replicas at all the times. Data encryption is the best method to ensure data security. A simple way to encrypt data is to create multiple replicas making them look unique and differentiable by using probabilistic encryption schemes. Probabilistic encryption creates different ciphertexts each time the same message is encrypted using the same key. Thus, our scheme uses homomorphic probabilistic encryption to encrypt data and create distinct replicas/copies of the data file. We use Boneh–Lynn–Shacham signature scheme (BLS) signatures [5] to create constant amount of metadata for any number of replicas. Probabilistic encryption encrypts all the replicas with the same key. Therefore, in our scheme the data owner will have to share just one decryption key with the authorized users and need not worry about CSP granting access to any of the replicas to the authorized users. The homomorphic property of the encryption scheme helps in efficient file updates. The data owner has to encrypt the difference between the updated file and the old file and send it to the cloud, which updates all the replicas by performing homomorphic addition on the file copies. Any authenticated data structure such as Merkle Hash Trees (MHT) or Skiplist can be used with our scheme to ensure that the cloud uses the right file blocks for data integrity verification. Barsoum and Hasan [4] shows how to use MHT for data integrity verification. Barsoum and Hasan [4] also shows another approach to ensure that the cloud uses the right file blocks by using a Map Version Table. Their results show that Map Version Table incurs less overhead compared to MHT. Both these approaches can be extended to our scheme. However, the ways to efficiently manage authenticated data structures and Map Version Table in the cloud is not within the scope of this paper. RSA signatures used in [9] can also be used with DMR-PDP scheme for creating data tags. We compare the performance of BLS and RSA signatures and discuss the benefits of using BLS signatures over RSA singatures. We identify the possible attacks the CSP can use to cheat the data owner and provide a security analysis of the proposed protocol against these attacks. Efficiency of the protocol has been experimentally tested and the results demonstrate that our protocol is more efficient than the scheme in [4].

We also extended the DMR-PDP scheme to support a basic file versioning system. The advantage of a file version system is that the data owner can review the changes done to the file and also retrieve the older versions of the file. The cloud computing solutions like Dropbox, Google Drive provide basic file version control system in addition to data storage service. The client-server file versioning models like SVN offer much more features when compared to what these cloud solutions provide. All these solutions use Delta compression technique where only the difference between the original file and the updated file is propagated and stored on the server. The differences are recorded in discrete files called "deltas" or "diffs". To retrieve any particular version of the file, the change or delta stored on the server is merged with the base version. This reduces the bandwidth and storage space on the server. In our present setting, the data owner uses a homomorphic probabilistic encryption scheme to encrypt and generates multiple encrypted replicas of the file. The cloud stores the replicas across multiple servers. The challenge is to store multiple replicas of the file and maintain the file updates as deltas to support a basic file versioning system when the files are encrypted. To address this, we propose multiple replica file version control system (MRFVCS) as an extension to the DMR-PDP scheme, which supports encrypted file version control when the data is replicated and the data owner can still use the DMR-PDP scheme to verify the integrity of the data. We implemented the MRFVCS scheme and our experiments show that it is efficient maintaining versions and needs only a little data storage on the data owner side .

Organization: The rest of the paper is organized as follows. Overview of the related work is provided in Sect. 2 followed by the a detailed description of our scheme in Sect. 3, and security analysis in Sect. 4. MRFVCS is discussed in Sect. 5 followed by Experimental results in Sect. 6 and conclusion in Sect. 7.

2 Related work

Ateniese et al. [1] were the first to define the PDP model for ensuring the possession of files on untrusted storages. They made use of RSA-based homomorphic tags for auditing outsourced data. However, dynamic data storage and multiple replica system are not considered in this scheme. In their subsequent work [2] and [21], they proposed a dynamic version which supports very basic block operations with limited functionality, and does not support block insertions. In [23], Wang et al. considered dynamic data storage in a distributed scenario, and proposed a challenge-response protocol which can determine data correctness as well as locate possible errors. Similar to [21], only partial support for dynamic data operation is considered. Erway et al. [12] extended the PDP model in [1] to support dynamic updates to stored data files using rank-based authenticated skip lists. However, the efficiency of their scheme remains unclear and these schemes hold good only for verifying a single copy. Wang et al. [22] used MHT for data integrity verification and their scheme supports dynamic data operations, but in their scheme data is not encrypted and works only for a single copy. Hao et al. [15] proposed a scheme that supports both the dynamic data operations and public verifiability. Public verifiability allows anyone, not necessarily the data owner, to verify the integrity of data stored in the cloud by running the challenge-response protocol. Their scheme also does not consider data encryption and their scheme cannot be extended to suit the scenario where multiple data copies are stored.

Curtmola et al. [9] proposed multiple-replica PDP (MR-PDP) scheme wherein the data owner can verify that several copies of a file are stored by a storage service provider. In their scheme, distinct replicas are created by first encrypting the data and then masking it with the randomness generated from a pseudo-random function (PRF). The randomized data is then stored across multiple servers. The scheme uses RSA signatures for creation of tags. But, their scheme did not address how the authorized users of the data can access the file copies from the cloud servers noting that the internal operations of the CSP are opaque and do not support dynamic data operations. In [19], Shacham and Waters proposed a scheme to verify the integrity of a single file copy. They defined their goal as to provide data availability and PDP guarentee to the data owner. They state that their scheme can be extended to support multiple file replicas. In their scheme, the files are erasure encoded before storing on the cloud. This scheme works well with static files and does not hold good for dynamic files. To update an erasure encoded file, the data owner has to download the entire file, decode it, update the required file blocks, erasure encode the updated file and then store it on the cloud. This is clearly an overhead to download entire file to update only a few file blocks and erasure encode the entire file after updating it. When their scheme is extended to a multiple replica scenario, the file update operation becomes more expensive since all the replicas should be downloaded, updated, erasure encoded and all the replicas should be uploaded to the cloud. Also, the file tags should be constructed for each file replica. We use the idea of homomorphic linear authenticator (HLA) and the approach of providing PDP guarantee used in [19] but our file construction, tag construction, proofs completely varies from the approach used in [19]. Our goal also differs from the goal defined in [19] as we also aim to provide data confidentiality along with data availability and PDP guarantee to the data owner. The protocol provided by Bowers et al. in [6] provides resilience to drive crashes and failures but does not deal with file updates and does not work well with multiple geolocation setting. This protocol requires all disks to be in the same place. In this scheme the files are erasure encoded before storing it on the cloud, and has the same overhead as [19]. Shacham and Waters [19] and Bowers et al. [6] does not deal with the scenario where the files are shared with multiple users. In this scenario it becomes a burden for the data users to download the entire file and remove the erasure encoding, even if they need only a few file blocks. Barsoum et al. [4] proposed creation of distinct copies by appending replica number to the file blocks and encrypting it using an encryption scheme that has strong diffusion property, e.g., AES. Their scheme supports dynamic data operations but during file updates, the copies in all the servers should be encrypted again and updated on the cloud. This scheme suits perfectly for static multiple replicas but proves costly in a dynamic scenario. BLS signatures are used for creating tags and authenticated data structures like MHT are used to ensure that the right file blocks are used during verification. Authorized users of the data should know random numbers in [9] and replica number in [4] to generate the original file.

The goal of DMR-PDP scheme differs significantly from the goal defined in the earlier reported schemes. DMR-PDP aims to provide data confidentiality, data integrity along with PDP guarantee to the data owner. Providing PDP guarantee in an encrypted multi-replica scenario is quite challenging. The novelty in our approach comes with providing data security and also to provide low cost updates to the file while preserving the PDP guarantee. Our scheme also can be easily extended to support multi versioning.

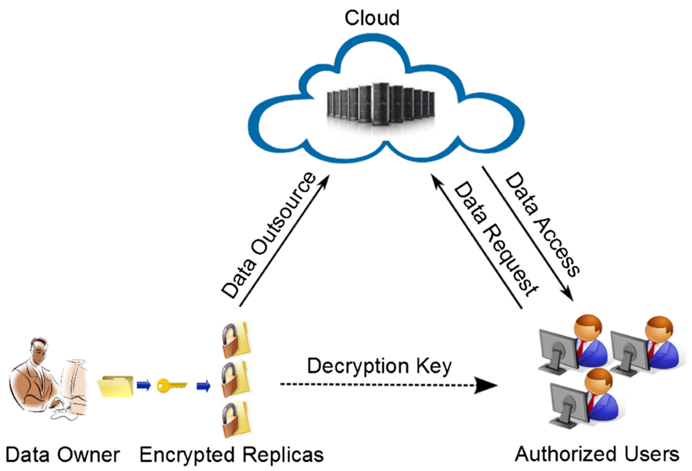


Fig. 1 Cloud computing data storage model

3 Dynamic multi-replica provable data possession (DMR-PDP) scheme

The cloud computing model considered in this work consists of three main components as illustrated in Fig. 1: (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 the cloud servers 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 keys with the data owner.

3.1 Problem definition and design goals

More recently, many data owners relieve their burden of local data storage and maintenance by outsourcing their data to a CSP. CSP undertakes the data replication task in order to increase the data availability, durability and reliability but the customers have to pay for using the CSPs storage infrastructure. On the other hand, the cloud customers should be convinced that the (1) CSP actually possesses all the data copies as agreed upon, (2) integrity of these data copies are maintained, and (3) the owners are able to update the data that they are paying for. Therefore, in this paper, we address the problem of securely and efficiently creating multiple replicas of the data file of the owner to be stored over untrusted CSP and then auditing all these copies to verify their completeness and correctness. Our design goals are summarized below:

1) Dynamic multi-replica provable data possession (DMR-PDP) protocols should efficiently and securely provide the owner with strong evidence that the CSP is in possession of all the data copies as agreed upon and that these copies are intact.
2) Allowing the users authorized by the data owner to seamlessly access a file copy from the CSP.

3) Using only a single set of metadata/tags for all the file replicas for verification purposes.
4) Allowing dynamic data operation support to enable the data owner to perform block-level operations on the data files while maintaining the same level of data correctness assurance.
5) Enabling both probabilistic and deterministic verification guarantees.

3.2 Preliminaries and notations

In this section, we provide details of the Bilinear mapping and Paillier encryption schemes used in our present work.

1) Assume that F, a data file to be outsourced, is composed of a sequence of m blocks, i.e., $F = \{b_1, b_2, \ldots, b_m\}$ where $b_i \in \mathbb{Z}_{\mathbb{N}}$, where $\mathbb{Z}_{\mathbb{N}}$ is the set of all residues when divided by N and N is a public key in Paillier scheme.
2) Let F_i represent the file copy i. So $F_i = \{b_{i1}, b_{i2}, \ldots, b_{im}\}$, where b_{ij} represents file block b_j of file copy i.
3) BLS signatures: BLS signatures are short homomorphic signatures that use the properties of bilinear pairings on certain elliptic curves. These signatures allow concurrent data verification, where multiple blocks can be verified at the same time.
4) Bilinear map/pairing: Let G_1, G_2, and G_T be cyclic groups of prime order a. Let u and v be generators of G_1 and G_2, respectively. A bilinear pairing is a map $e : G_1 \times G_2 \rightarrow G_T$ with the following properties:
   - Bilinear: $e(u_1u_2, v_1) = e(u_1, v_1) \cdot e(u_2, v_1)$, $e(u_1, v_1v_2) = e(u_1, v_1) \cdot e(u_1,v_2)$ $\forall$ $u_1, u_2 \in G_1$ and $v_1, v_2 \in G_2$
   - Non-degenerate: $e(u, v) \neq 1$
   - There exists an efficient algorithm for computing e
   - $e(u_1^x, v_1^y) = e(u_1, v_1)^{xy}$ $\forall$ $u_1 \in G_1$; $v_1 \in G_2$, and $x, y \in Z_a$
5) H(.) is a map-to-point hash function: $\{0, 1\}^* \rightarrow G_1$.
6) Homomorphic encryption: A homomorphic encryption scheme has the following properties.
   - $E(m_1 + m_2) = E(m_1) +_h E(m_2)$ where $+_h$ is a homomorphic addition operation.
   - $E(k{*}m) = E(m)^k$.

   where E(.) represents a homomorphic encryption scheme and m, m_1, m_2 are messages that are encrypted and k is some random number.
7) Paillier encryption: Paillier cryptosystem is a homomorphic probabilistic encryption scheme. The steps are as follows.
   - Compute $N = p * q$ and $\lambda = LCM(p-1, q-1)$, where p, q are two prime numbers.
   - Select a random number g such that its order is a multiple of N and $g \in Z_N^{2*}$.
   - Public key is (N, g) and secret key is λ, where $N = p*q$.
   - Ciphertext for a message m is computed as $c = g^m r^N \bmod N^2$ where r is a random number and $r \in Z_N^*$, $c \in Z_N^{2*}$ and $m \in Z_N$.
   - Plain text is obtained by $m = L(c^\lambda \bmod N^2) * (L(g^\lambda \bmod N^2))^{-1} \bmod N$.
8) Properties of public key g in Paillier Scheme

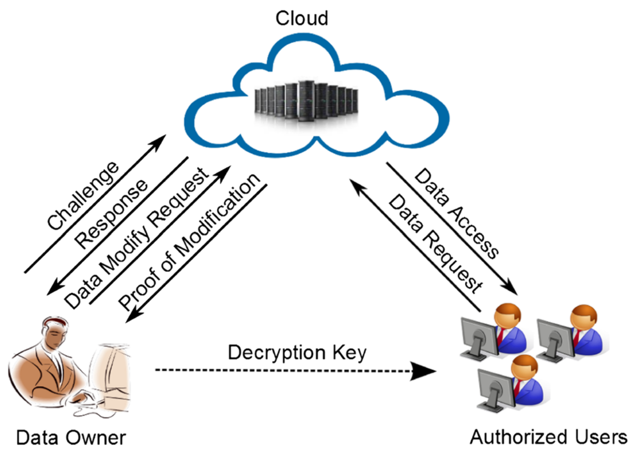


Fig. 2 DMR-PDP scheme

- $g \in Z_N^{2*}$.
- If g = (1 + N) mod N^2, it has few interesting properties
  a) Order of the value (1 + N) is N.
  b) $(1 + N)^m \equiv (1 + mN) \bmod N^2$. (1 + mN) can be used directly instead of calculating $(1 + N)^m$. This avoids the costly exponential operation during data encryption.

3.3 DMR-PDP construction

In our approach, the data owner creates multiple encrypted replicas and uploads them on to the cloud. The CSP stores them on one or multiple servers located at various geographic locations. The data owner shares the decryption key with a set of authorized users. In order to access the data, 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. The proposed scheme consists of seven algorithms: KeyGen, ReplicaGen, TagGen, Prove, Verify, PrepareUpdate and ExecUpdate. The overview of the communication involved in our scheme is shown in Fig. 2.

1) (pk, sk) ← KeyGen(). This algorithm is run by the data owner to generate a public key *pk* and a private key *sk*. The data owner generates five sets of keys.
   a) Key for data tags: This key is used for generating tags for the data. The data owner selects a bilinear map *e* and selects a private key $l \in Z_a$, where l is the private key and a is the order of cyclic groups G_1, G_2, and G_T. Public key is calculated as $y = v^l \in G_2$, where v is the generator of group G_2.
   b) Key for data: This key is used for encrypting the data and thereby creating multiple data copies. The data owner selects paillier public keys (N, g) with g = (1 + N) mod N^2 and secret key λ.
   c) PRF key for verification: The data owner generates a PRF key Key_{PRF} which generates s numbers. These s numbers are used in creating s copies of the data.

Each number is used in creating one data copy. Let $\{k_1, k_2, \ldots, k_s\} \in Z_N^*$ be the numbers generated by the PRF key. Key_{PRF} is maintained confidentially by the data owner and hence the *s* numbers used in creating multiple copies are not known to the cloud.

d) PRF key for Paillier encryption: The data owner generates a PRF key Key_{rand}, which is used for generating the random numbers used in Paillier encryption.

e) PRF key for Tag generation: The data owner generates a PRF key Key_{tag}, which is used in generation of tags.

2) $\{F_i\}_{1 \le i \le s} \leftarrow$ ReplicaGen (s, F). This algorithm is run by the data owner. It takes the number of replicas *s* and the file *F* as input and generates *s* unique differentiable copies $\{F_i\}_{1 \le i \le s}$. This algorithm is run by the data owner only once. Unique copies of each file block of file F is created by encrypting it using a probabilistic encryption scheme, e.g., Paillier encryption scheme. Through probabilistic encryption, encrypting a file block *s* times yields *s* distinct ciphertexts. For a file $F = \{b_1, b_2, \ldots, b_m\}$ multiple data copies are generated using Paillier encryption scheme as $F_i = \{(1+N)^{b_1}(k_i r_{i1})^N, (1+N)^{b_2}(k_i r_{i2})^N, \ldots, (1+N)^{b_m}(k_i r_{im})^N\}_{1 \le i \le m}$. Using Paillier's properties the above result can be rewritten as $F_i = \{(1+b_1N)(k_i r_{i1})^N, (1+b_2N)(k_i r_{i2})^N, \ldots, (1+b_mN)(k_i r_{im})^N\}_{1 \le i \le m}$, where *i* represents the file copy number, k_i represents the numbers generated from PRF key Key_{PRF} and r_{ij} represents random number used in Paillier encryption scheme generated from PRF key Key_{rand}. k_i is multiplied by the random number r_{ij} and the product is used for encryption. The presence of k_i in a block identifies which copy the file block belongs to. All these file copies yield the original file when decrypted. This allows the users authorized by the data owner to seamlessly access the file copy received from the CSP.

3) $\phi \leftarrow$ TagGen (sk, F). This algorithm is run by the data owner. It takes the private key sk and the file F as input, and outputs the tags ϕ. We use BLS signature scheme to create tags on the data. BLS signatures are short and homomorphic in nature and allow concurrent data verification, which means multiple data blocks can be verified at the same time. In our scheme, tags are generated on each file block b_i as $\phi_i = (H(F) \cdot u^{b_i N + a_i})^l \in G_1$ where $u \in G_1$, $H(.) \in G_1$ represents hash value which uniquely represents the file F and $\{a_i\}_{1 \le i \le m}$ are numbers generated from PRF key Key_{tag} to randomize the data in the tag. Randomization is required to avoid generation of same tags for similar data blocks. The data owner sends the tag set $\phi = \{\phi_i\}_{1 \le i \le m}$ to the cloud.

4) $P \leftarrow$ Prove (F, ϕ, *challenge*). This algorithm is run by the CSP. It takes the replicas of file F, the tags ϕ and *challenge* vector sent by the data owner as input and returns a proof *P* which guarantees that the CSP is actually storing *s* copies of the file F and all these copies are intact. The data owner uses the proof *P* to verify the data integrity. There are two phases in this algorithm:

a) **Challenge:** In this phase, the data owner challenges the cloud to verify the integrity of all outsourced copies. There are two types of verification schemes:

i) Deterministic—here all the file blocks from all the copies are used for verification.

ii) Probabilistic—only a few blocks from all the copies are used for verification. A PRF key is used to generate random indices ranging between 1 and *m*. The file blocks from these indices are used for verification. In each verification a percentage of the file is verified and it accounts for the verification of the entire file.

At each challenge, the data owner chooses the type of verification scheme he wishes to use. If the owner chooses the deterministic verification scheme, he generates one PRF key, Key_1. If he chooses the probabilistic scheme he generates two PRF keys, Key_1 and Key_2. PRF keyed with Key_1 generates c ($1 \leq \text{c} \leq \text{m}$) random file indices which indicates the file blocks that CSP should use for verification. PRF keyed with Key_2 generates s random values and the CSP should use each of these random numbers for each file copy while computing the response. The data owner sends the generated keys to the CSP.

b) **Response:** This phase is executed by the CSP when a challenge for data integrity verification is received from the data owner. Here, we show the proof for probabilistic verification scheme (the deterministic verification scheme also follows the same procedure). The CSP receives two PRF keys, Key_1 and Key_2 from the data owner. Using Key_1, CSP generates a set $\{\text{C}\}$ with c ($1 \leq \text{c} \leq \text{m}$) random file indices ($\{\text{C}\} \in \{1, 2, \ldots, \text{m}\}$), which indicate the file blocks that CSP should use for verification. Using Key_2, CSP generates 's' random values $\text{T} = \{\text{t}_1, \text{t}_2, \ldots, \text{t}_s\}$. The cloud performs two operations; One on the tags and the other on the file blocks.

i) Operation on the tags: Cloud multiplies the file tags corresponding to the file indices generated by PRF key Key_1.

$$\sigma = \prod_{j \in C} \left(H(F) \cdot u^{b_j N + a_j} \right)^l = \prod_{j \in C} H(F)^l \cdot \prod_{j \in C} u^{(b_j N + a_j) l}$$
$$= H(F)^{cl} \cdot u^{\left(N \sum_{j \in C} b_j + \sum_{j \in C} a_j \right) l}$$

ii) Operation on the file blocks: The cloud first takes each file copy and multiplies all the file blocks corresponding to the file indices generated by the PRF key Key_1. The product of each copy is raised to the power the random number generated for that copy by the PRF key Key_2. The result of the above operation for each file copy i is given by $(\prod_{j \in C} (1 + N)^{b_j} (k_i r_{ij})^N)^{t_i} \bmod \text{N}^2$. The CSP then multiplies the result of each copy to get the result

$$\mu = \prod_{i=1}^{s} \left(\prod_{j \in C} (1 + N)^{b_j} (k_i r_{ij})^N \right)^{t_i}$$
$$= \prod_{i=1}^{s} \left(\prod_{j \in C} (1 + N)^{b_j t_i} \prod_{j \in C} (k_i r_{ij})^{N t_i} \right)$$
$$= \prod_{i=1}^{s} \left((1 + N)^{t_i \sum_{j \in C} b_j} \prod_{j \in C} (k_i r_{ij})^{N t_i} \right)$$

$$= \left(\prod_{i=1}^{s}(1+N)^{t_i \sum_{j\in C} b_j}\right)\left(\prod_{i=1}^{s}\left((k_i)^{ct_i N}\prod_{j\in C}(r_{ij})^{Nt_i}\right)\right)$$

$$= \left((1+N)^{\sum_{i=1}^{s} t_i \sum_{j\in C} b_j}\right)\left(\prod_{i=1}^{s}(k_i)^{ct_i N}\right)\left(\prod_{i=1}^{s}\prod_{j\in C}(r_{ij})^{Nt_i}\right)$$

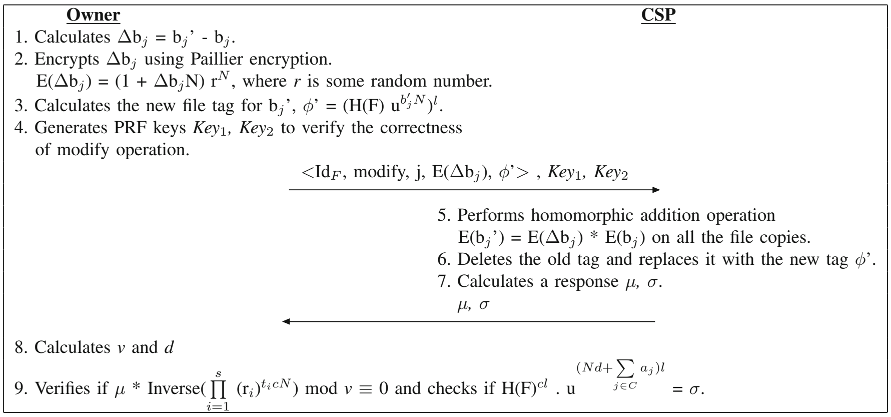
Using properties of Paillier scheme, the above equation can be rewritten as

$$\mu = \left(1 + N\sum_{i=1}^{s}(t_i)\sum_{j\in C}(b_j)\right)\left(\prod_{i=1}^{s}(k_i)^{Nct_i}\right)\left(\prod_{i=1}^{s}\left(\prod_{j\in C}(r_{ij})^{t_i N}\right)\right)$$

The CSP sends σ and μ mod N^2 values to the data owner.

5) $\{1, 0\} \leftarrow$ Verify (pk, P). This algorithm is run by the data owner. It takes as input the public key pk and the proof P returned from the CSP, and outputs 1 if the integrity of all file copies is correctly verified or 0 otherwise. After receiving σ and μ values from the CSP, the data owner does the following:
   a) calculates v = $(\prod_{i=1}^{s} (\mathrm{k}_i)^{t_i c N})$ and d = Decrypt(μ) * Inverse($\sum_{i=1}^{s}$ t_i). This can be calculated from the values generated from Key_{rand}, Key_{PRF} and the value c.
   b) checks if μ * Inverse($\prod_{i=1}^{s}$ $(\mathrm{r}_i)^{t_i c N}$) mod v $\equiv 0$. This ensures that the cloud has used all the file copies while computing the response.
   c) checks if $(\mathrm{H(F)})^c$ $\mathrm{u}^{dN+\sum_{j\in C} a_j})^l = \sigma$. The random numbers in the tag are generated from PRF key Key_{tag}. This ensures that the CSP has used all the file blocks while computing the response. If options b and c are satisfied, it indicates that the data stored by the owner in the cloud is intact and the cloud has stored multiple copies of the data as agreed in the service level agreement.
6) *Update* $\leftarrow$ PrepareUpdate (). This algorithm is run by the data owner to perform any operation on the outsourced file copies stored by the remote CSP. The output of this algorithm is an *Update* request. The data owner sends the *Update* request to the cloud that will be of the form $\langle \mathrm{Id}_F$, BlockOp, j, b'_i, $\phi' \rangle$, where Id_F is the file identifier, BlockOp corresponds to block operation, j denotes the index of the file block, b'_i represents the updated file blocks and ϕ' is the updated tag. BlockOp can be data modification, insertion or delete operation.
7) $(\mathrm{F}', \phi') \leftarrow$ ExecUpdate (F, ϕ, *Update*). This algorithm is run by the CSP where the input parameters are the file copies F, the tags ϕ, and *Update* request (sent from the owner). It outputs an updated version of all the file copies F' along with updated signatures ϕ'. After any block operation, the data owner runs the challenge protocol to ensure that the cloud has executed the operations correctly. The operation in *Update* request can be modifying a file block, inserting a new file block or deleting a file block.

a) **Modification:** Data modification is one of the most frequently used dynamic operations. The data modification operation in DMR-PDP scheme is shown in Fig. 3.



Owner **CSP**

1. Calculates $\Delta b_j = b_j' - b_j$.
2. Encrypts Δb_j using Paillier encryption.
   $E(\Delta b_j) = (1 + \Delta b_j N)\, r^N$, where r is some random number.
3. Calculates the new file tag for b_j', $\phi' = (H(F)\, u^{b'_j N})^l$.
4. Generates PRF keys Key_1, Key_2 to verify the correctness of modify operation.

$$\xrightarrow{<Id_F,\ modify,\ j,\ E(\Delta b_j),\ \phi'>\ ,\ Key_1,\ Key_2}$$

5. Performs homomorphic addition operation
   $E(b_j') = E(\Delta b_j) * E(b_j)$ on all the file copies.
6. Deletes the old tag and replaces it with the new tag ϕ'.
7. Calculates a response μ, σ.

$$\xleftarrow{\mu,\ \sigma}$$

8. Calculates v and d
9. Verifies if μ * Inverse($\prod_{i=1}^{s} (r_i)^{t_i c N}$) mod $v \equiv 0$ and checks if $H(F)^{cl} \,.\, u^{(Nd+\sum_{j\in C} a_j)l} = \sigma$.

Fig. 3 Block modification operation in the DMR-PDP scheme

b) **Insertion:** In the block insertion operation, the owner inserts a new block after position j in a file. If the file F had m blocks initially, the file will have $m + 1$ blocks after the insert operation. The file block insertion operation is shown in Fig. 4.

c) **Deletion:** Block deletion operation is opposite of the insertion operation. When one block is deleted, indices of 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 $\langle Id_F$, delete, j, null, null$\rangle$ to the cloud. Upon receiving the request, the cloud deletes the tag and the file block at index j in all the file copies.

3.4 Using RSA signatures

DMR-PDP scheme works well even if RSA signatures are used instead of BLS singatures. The complete DMR-PDP scheme using RSA signatures is shown in Fig. 5.

4 Security analysis

In this section, we present a formal analysis of the security of our proposed scheme. The data owner encrypts the files and stores them on the cloud which is untrusted and so we identify the cloud as the main adversary in this scheme. The scheme is secure only if it does not let the cloud cheat the data owner by deleting file blocks and still pass the challenge/response phase initiated by the data owner.

- Security against forging the response by the adversary : In our scheme, the security against forging the response by the adversary follows the proof of PDP guarantee provided in [19]. In the challenge phase, the data owner sends to the CSP, two PRF keys— Key_1, Key_2 and a parameter ‘c’ which indicates number of file blocks he wishes to verify. DMR-PDP scheme provides flexiblity to the data owner to send different ‘c’ and PRF keys in each challenge phase to the CSP. This ensures that

Owner **CSP**

1. Encrypts the new file block s times
2. Creates tag ϕ for the new file block
3. Generates PRF keys Key_1, Key_2 to verify the correctness of insert operation.

$$\xrightarrow{<\mathrm{Id}_F,\ \text{insert, j},\ s\ \text{file blocks},\ \phi>\ ,\ Key_1,\ Key_2}$$

4. Inserts the new file block at location j
5. Stores the new tag ϕ.
6. Calculates a response μ, σ.

$$\xleftarrow{\mu,\ \sigma}$$

7. Calculates v and d
8. Verifies if μ* Inverse($\prod_{i=1}^{s}$ $(\mathrm{r}_i)^{t_i cN}$)mod $v \equiv 0$ and checks if $\mathrm{H(F)}^{cl}$. $\mathrm{u}^{(Nd+\sum_{j\in C} a_j)l}$ = σ.

Fig. 4 Block insertion operation in the DMR-PDP scheme

Preliminaries

1. Data owner selects two prime numbers p, q
2. $N = pq$ is the RSA modulus
3. g is the generator of QR_N (QR_N is the set of quadratic residues modulo N)
4. Publick key is (N, g) and secret key is (p, q)
5. Data owner encrypts the file blocks s times
6. Data owner generates tags T_i for each file block b_i, where $T_i = g^{b_i}$ mod N
6. The data file and the data tags are sent to the cloud.

Owner **CSP**

1. Generates PRF keys Key_1, Key_2 and sends them to the cloud to verify data integrity

$$\xrightarrow{\mathrm{Id}_F,\ Key_1,\ Key_2}$$

2. Calculates μ
3. Calculates response using data tags

$$\sigma = \prod_{i=1}^{s} \mathrm{g}^{b_i} \bmod \mathrm{N}$$

$$\xleftarrow{\mu,\ \sigma}$$

4. Calculates v and d
5. Verifies if μ * Inverse($\prod_{i=1}^{s}$ $(\mathrm{r}_i)^{t_i cN}$) mod $v \equiv 0$ and checks if g^d mod N = σ.

Fig. 5 DMR-PDP scheme using RSA signatures

the response generated by the CSP will not be the same for each challenge sent by the data owner. This eliminates any opportunity for the CSP to forge the response without actually calculating it.

- Security against deletion of file blocks with same value: When files are split into blocks there are chances that few of the split blocks have the same value. The data tags generated will be the same for file blocks with same value. Though the file blocks have same value their ciphertexts will not have the same value since Paillier cryptosystem is probabilistic. Though the file blocks are encrypted and encrypted blocks do not have the same value, the cloud can identify the file blocks with same value by identifying tags with the same value. Cloud can cheat the user by just storing one block and deleting data blocks that have the same file tag. To avoid it, DMR-PDP scheme randomizes the data before constructing the tags. The data in the tags are added with random numbers generated from PRF key Key_{tag}. So, even if the data tag values are the same, the underlying data file block values will not be the same. For file blocks $b_i = b_j$

$$Tag(b_i) = \left(H(F) \cdot u^{b_i N + a_i}\right)^l$$
$$Tag(b_j) = \left(H(F) \cdot u^{b_j N + a_j}\right)^l$$

where a_i, a_j are the random numbers generated from PRF key Key_{tag}. Though the data blocks are the same, the generated tags will differ in value.

- Security of Paillier Cryptosystem when g = N + 1: In DMR-PDP scheme the multiple replica preparation algorithm uses Paillier encryption. So, the security of DMR-PDP scheme depends on the security of the Paillier cryptosystem. Choi et al. [8] discusses the one-wayness and semantic security of using the public key g = N + 1 in Paillier cryptosystem. According to [8], the public key g = N + 1 satisfies the semantic security and one-wayness property. The Paillier cryptosystem is as intractable as factoring the modulus N, if the public key g can be generated using the public information N. Damgard and Jurik [10] and Catalano et al. [7] use this observation g = N + 1 in their modified constructions of Paillier encryption. So, the DMR-PDP scheme is also secure.

  The security of DMR-PDP scheme derives from the combination of the security schemes mentioned above. While Pallier encryption addresses the data security concern, the data availability issue is addressed by the PDP scheme The data owner encrypts the data using Paillier encryption scheme before storing it on the cloud. The data security is ensured by the Paillier encryption scheme. For the PDP guarantee, the arithmetic operations are done on the Paillier encrypted data to generate μ and the tags constructed using BLS signatures to generate σ. The security of these arithmatic operations are shown in [8] and [5] respectively and the security of PDP approach is outlined in [1]. The security of DMR-PDP scheme is obtained thus by combining these security schemes.

5 Multiple replica file version control system (MRFVCS)

MRFVCS is an extension to DMR-PDP scheme to support a basic file versioning system. The data owner encrypts the data, creates multiple replicas and stores them on the cloud. When the data is updated, the data files are not updated directly rather the updates are maintained as deltas. With MRFVCS the data owner can still use the DMR-PDP scheme to verify that the cloud stores multiple replicas and the deltas intact.

5.1 MRFVCS construction

The data owner divides the file into multiple file blocks and generates unique multiple replicas and data tags for the file blocks. Unique multiple replicas of the file blocks are generated using the homomorphic probabilistic encryption scheme and the data tags for the file blocks are generated using the BLS signatures as discussed in Sect. 3.3. The file block replicas and the data tags for the file blocks are sent to the cloud. These encrypted replicas of the file blocks represent the base version of the un-encrypted file blocks. Any modification done to the latest version of un-encrypted file blocks will result in a new version of the file blocks. The new version of the file blocks are not stored directly on the cloud, and instead, the deltas are stored. Delta is calculated as the difference between the un-encrypted new version of the file block and the un-encrypted base version of the file block. When a particular version of the file blocks are needed, the data owner requests the cloud to merge the delta blocks with the base version of the file blocks to get the required version of the file blocks. The data owner uses a file version table to track the versions of the file blocks. The table is a small data structure stored on the verifier side to validate the integrity and consistency of all files and its versions stored by the CSP. New versions of the file blocks are generated when the data owner performs an update operation on the file blocks. The data update operation includes inserting new file blocks or modifying or deleting a few file blocks. Delta blocks are generated only when the update operation is 'modify'. Once the update operation is done, the file version table is updated by the data owner. The file version table is maintained only on the data owner side, which also helps the data owner to hide the details of update operations from the CSP. The file version table consists of five columns: Block Number (BN), Delta Block Number (DBN), File Version (FV), Block Version (BV), Block Operation (BO). The BN acts as an indexing to the file blocks. It indicates the physical position of a block in a data file. The DBN is an indexing to the delta block. If delta does not exist, the value is stored as '–'. The FV indicates the version of the entire file and BV indicates the version of the file block. The BO indicates the operation done on the file block. The maximum value of FV gives the latest version of the file and the maximum value of BV for a particular BN gives the latest version of that particular file block. If no entry of a file block number is found in the file version table, it means no update operations are done on the base version of the file block and the file block in the base version and the latest version of the file are same. When a file block with block number B, file version V, and file block version Y is modified, the data owner may choose to change the version of entire file to V+1 or keep the file blocks under the same version. For both these cases, a new entry is made in the file version table by the data owner. In first case, the table entry will be ⟨B, –, V+1, 0, Modify⟩ and for the seond case, the table entry will be ⟨B, –, V, Y+1, Modify⟩. The proposed scheme consists of seven algorithms : Keygen, ReplicaGen, TagGen, Prove, Verify, PrepareUpdate, ExecUpdate, FileVersionRequest, FileVersionDeliver.

1) (pk,sk) $\leftarrow$ KeyGen(). Along with the keys described in Sect. 3.3, the data owner generates a PRF key Key_{data} which is used to randomize the file blocks before encryption.

2) $\{F'_{B_i}\}_{1\le i\le s}$ ← ReplicaGen (s, F_B). This algorithm is run by the data owner and it slightly differs from the algorithm described in Sect. 3.3. In this algorithm, the data owner randomizes the data before generating multiple replicas. File blocks are randomized using the random numbers generated from the PRF key Key_{data}. For a file $F_B = \{b_1, b_2, \ldots, b_m\}$, the randomized file will be F'_B, which is $\{b_1 + x_1, b_2 + x_2, \ldots, b_m + x_m\}$ where F_B represents the base version of the file and $\{x_j\}_{1\le j\le m}$ are the random numbers generated using Key_{data}. The data owner uses Paillier encryption, a homomorphic probabilistic encryption scheme, to create s replicas of the file F_B. So, $F'_{B_i} = \{(1+(b_1 + x_1)N)(k_i r_{i1})^N, (1+(b_2 + x_2)N)(k_i r_{i2})^N, \ldots, (1+(b_m + x_m)N)(k_i r_{im})^N\}_{1\le i\le m}$, where i represents the file copy number, k_i represents the numbers generated from PRF key Key_{PRF} and r_{ij} represents random number used in Paillier encryption scheme generated from PRF key Key_{rand} (discussed in Sect. 3.3). The delta files are generated by the data owner when the latest version of the file blocks are updated. The delta files are not encrypted. So, to increase security, the data blocks are randomized and then encrypted. When the delta files are generated, the data owner randomizes the delta files with the same random numbers used for randomizing the data blocks. The FileVersionDeliver algorithm explains how the cloud delivers a particular version of a file in this randomized scenario.
3) ϕ ← TagGen (sk, F). This algorithm is run by the data owner and BLS signatures are used to generate the data tags. The details of this algorithm are same as discussed in Sect. 3.3.
4) P ← Prove (F_B, F_Δ, ϕ, *challenge*). This algorithm is run by the CSP. It takes the replicas of file F_B, all delta files F_Δ, tags ϕ and the *challenge* vector sent by the data owner as input and returns a proof P. Proof P guarantees that the CSP is actually storing s copies of the file F_B and all the delta files F_Δ. The data owner uses proof P to verify data integrity. The details of this algorithm are the same as discussed in Sect. 3.3.
5) $\{1, 0\}$ ← Verify (pk, P). This algorithm is run by the data owner. It takes as input public key pk and the proof P returned from the CSP, and outputs 1 if the integrity of all file copies is correctly verified or 0 otherwise. The details of this algorithm are same as discussed in Sect. 3.3.
6) *Update* ← PrepareUpdate (). This algorithm is run by the data owner to perform any operation on the outsourced file copies stored by the remote CSP. The output of this algorithm is an *Update* request. The data owner sends the *Update* request to the cloud that will be of the form $\langle Id_F, \text{BlockOp}, j, b'_i, \phi'\rangle$, where Id_F is the file identifier, BlockOp corresponds to block operation, j denotes the index of the file block, b'_i represents the updated file blocks and ϕ' is the updated tag. BlockOp can be data insertion or modification or delete operation.
   a) Insertion: An insert operation on any version 'V' of the file F_V means inserting new file blocks in the file. The data owner decides the version of the file to which the new file blocks belong, either to current file version V or to next file version V+1. If the new file blocks are added to the file version 'V+1', then 'V+1' will be the new version of the file. A new entry is made in the file

version table as ⟨BN, – , V or V+1, 0, Insert⟩. Since there are no delta blocks, the DBN value is '–' and since the file blocks are new, the BV value is 0.

b) Modification: Modification is done on the latest version of the file blocks. The data owner identifies the block numbers of the file blocks that he wishes to modify and searches the file version table for block numbers. If no entry is found for a particular block number, the file blocks from the base version are downloaded from the cloud. If an entry is found, then the latest version of the file block is identified and downloaded from the cloud, the file blocks from the base version and the delta blocks associated with the latest version. The downloaded blocks are decrypted and added with the delta to get the latest version of the file blocks. Modify operation is done on the plain text to get the updated plain text. The data owner calculates the new delta as the difference between the updated plain text and the plain text belonging to the base version. Delta is then randomized and then sent to the cloud. Randomization is required in order to not reveal the delta value to the cloud. Let $\mathrm{M} = \{\mathrm{b}_i\}$ where $1 \le \mathrm{i} \le \mathrm{s}$ be the set of file blocks before the update operation and $\mathrm{M}' = \{\mathrm{b}'_i\}$ where $1 \le \mathrm{i} \le \mathrm{s}$ be the file blocks after the update operation. Deltas $\Delta\mathrm{M}$ are calculated as $\{\mathrm{b}'_i - \mathrm{b}_i\}$ where $1 \le \mathrm{i} \le \mathrm{s}$. Deltas are randomized using random numbers generated from PRF key Key_{data}. So, $\Delta\mathrm{M} = \{\mathrm{b}'_i - \mathrm{b}_i + \mathrm{N} - \mathrm{x}_i\}$ where $1 \le \mathrm{i} \le \mathrm{s}$. $\Delta\mathrm{M}$ values are sent to the cloud.

c) Deletion: A delete operation on any version of the file 'V' means deleting few file blocks from the file. The data owner can delete the file blocks from the current version V or delete the file blocks in the next version of the file which is V+1. The data owner makes an entry in the file version table ⟨BN, – , V or V+1, 0, Delete⟩. The result of the delete operation is just an entry in the file version table while the cloud does not know anything about the delete operation.

7) $(\mathrm{F}', \phi') \leftarrow$ ExecUpdate (F, ϕ, *Update*). This algorithm is run by the CSP where the input parameters are the file copies F, the tags ϕ, and *Update* request (sent by the data owner). It outputs new file copies F′ along with updated signatures ϕ'. After any block operation, the data owner runs the challenge protocol to ensure that the cloud has executed the operations correctly. The operation in *Update* request can be modifying a file block or inserting a new file block. The data owner does not send any delete requests to the cloud and so no data blocks will be deleted.

   a) Insertion: New file blocks sent by the data owner are added to the file F.
   b) Modify: Delta values sent by the data owner are stored on the cloud.

8) FileVersionRequest: The data owner identifies the file blocks of the required version by checking the file version table and sends a request to the cloud with the block numbers. The request will be of the form ⟨BN, DBN⟩. The DBN is required to get to the required version of the file. If there is no DBN entry in the file version table, then the request will be ⟨BN, – ⟩.

9) FileVersionDeliver: For all the file block numbers with DBN value '–' in the FileVersionRequest, the base version of the file is delivered to the data owner. If DBN has a delta file block number, then the cloud encrypts the deltas with the public key and does a homomorphic addition operation on the base version of

the file blocks to get the file blocks of the version requested by the data owner. Let $\Delta M = \{b'_i - b_i + N - x_i\}$ where $1 \leq i \leq s$, be the deltas associated with the file blocks of the corresponding file version requested by the data owner. The encrypted deltas $E(\Delta M)$ is given by $\{(1+(b'_i - b_i + N - x_i)N)(r)^N\}$, where $r \in Z_N{}^*$ is some random number. The cloud then performs a homomorphic addition operation on the requested file blocks on the base version of the file.

$$\begin{aligned} E(F_v) &= E(F_b) * E(\Delta M_v). \\ &= \left\{(1 + (b_i + x_i)N)(k_i r_i)^N\right\} \\ &\quad * \left\{(1 + (b'_i - b_i + N - x_i)N)(r)^N\right\}. \\ &= \left\{(1 + (b'_i N)(k_i r_i r)^N\right\}. \end{aligned}$$

The file blocks obtained after the homomorphic addition represents the encrypted file blocks of the version requested by the data owner. The encrypted file blocks are sent to the data owner and the data owner decrypts the file blocks to get the version he requested.

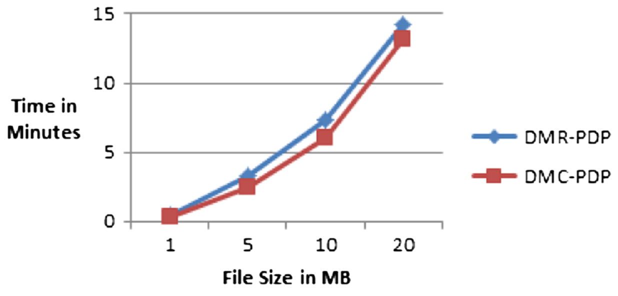
6 Implementation and experimental results

We implemented our scheme and the protocols in C language. We conducted several experiments using the local CentOS server as well as EC2 cloud instnaces with different configurations. We measured the computation time for various operations for both the CSP and the user. In addition, we also measured latency in terms of communication cost. Varying file sizes were considered in these experiments.

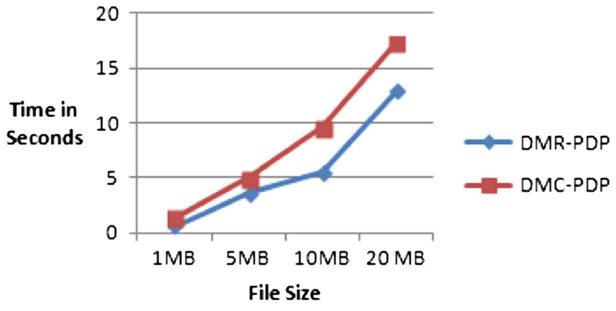
To start with, we conducted several experiments on a system with an Intel(R) Xeon (R) 2.67 GHZ processor and 11 GB RAM running CentOS 6.3. In our implementation, we use PBC library version 0.5.11. To achieve 80-bit security parameter, the curve group with 160-bit group order is selcted and the size of modulus N is 1,024 bits. We utilize the Barreto–Naehrig (BN) [3] curve defined over prime field GF(p) with |p| = 160 and embedding degree = 12. The data tags generated are points on this curve and a point on this curve can be represented by 160 bits. We use SHA algorithm for computing file hash, and PBC library provides functions to represent the hash values as a point on the curve. The data owner will have to store three PRF keys of size 128 bits, one secret key for data tags of size 128 bits and one secret key for data encryption of size 1024 bits. The communication cost for each phase incurred in this protocol is shown

Table 1 DMR-PDP communication cost

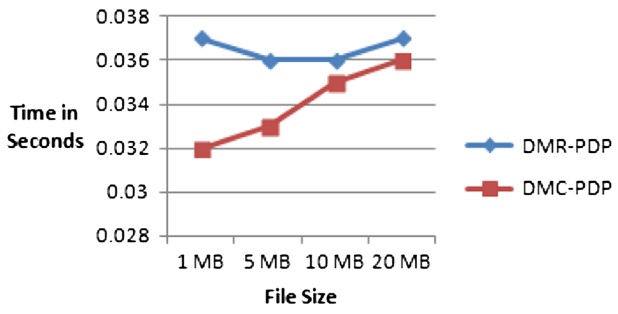
Phase	Cost	Data	From	To
Challenge	256 bits	Key_1, Key_2, c	Owner	Cloud
Verification	2048 + 160 bits	μ, σ	Cloud	Owner
Update	2048 + 160 bits	b'_i, ϕ'	Owner	Cloud



(a) Data setup time



(b) CSP computation time



(c) User computation time

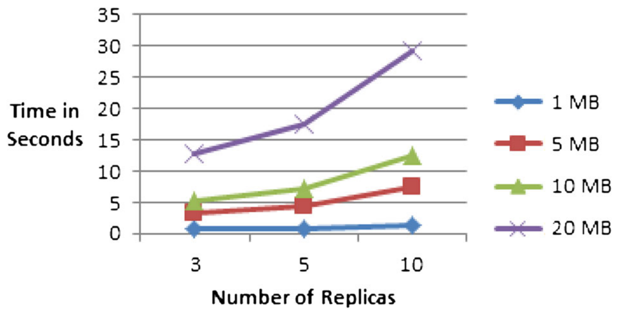
Fig. 6 Computation time comparison

in Table 1. Here, we compare the performance of the DMR-PDP scheme proposed in this paper with that of the DMC-PDP scheme proposed in [4]. The 1024 bit modulus used for Paillier encryption in this paper is comparable in terms of security to 128 bit AES encryption used in [4]. Figure 6 shows the Data setup, CSP and User computation times for both the schemes using files of sizes 1, 5, 10 and 20 MB with three replicas. The data setup is done only once on the data owner side. Figure 6a compares the time taken for data setup by DMR-PDP and DMC-PDP schemes. The data setup cost differs only in few minutes. The DMC-PDP scheme performs little faster than DMR-PDP scheme. The data setup in DMC-PDP scheme uses AES encryption which is faster than Paillier encryption, but the DMC-PDP scheme involves tag construction for all the replicas where as in DMR-PDP scheme only one set of tags are constructed for all the replicas. So, the computation time for data setup is almost same for both the schemes. The DMR-PDP scheme has lower CSP computation time compared to the DMC-PDP scheme whereas the User computation time for both the schemes differ only in a 1000th of a second. Both the schemes involve just the pairing operation in the user verification phase and hence similar User computation times. Performance of the DMR-PDP scheme is better than that of the DMC-PDP scheme and improves with increase in file size.

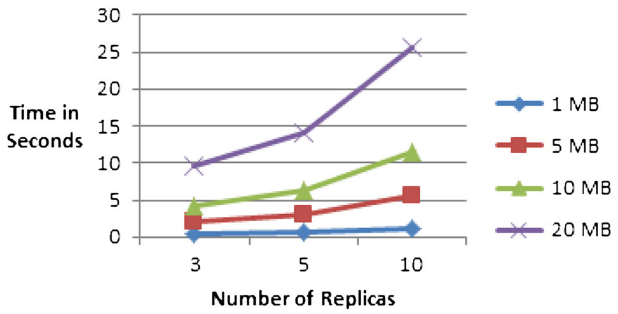
The CSP and User computation costs incurred during the response phase of the proposed scheme is depicted in Figs. 7a and 8a for 1, 5, 10 and 20 MB files with 2KB encrypted file block size. Figure 7a shows the computation time in seconds on the local CentOS server for different number of replicas. For the DMR-PDP scheme the User verification phase involves only one heavy pairing operation and the user computation time is independent of the number of replicas and the file size as shown in Fig. 8a. It has been reported in [9] 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 and is 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.

Figures 7b and 8b show the CSP and the User computation times on the local CentOS server when RSA signatures are used. For better security, we used N to be of size 1024 bits. For simplicity we use the same N used for data encryption as RSA modulus. From Fig. 7a, b, we notice that, the CSP computation time for BLS and RSA signatures is almost the same. Figure 8 shows the comparison of User computation times when BLS and RSA signatures are used. Since user verification of RSA signatures involves only exponentional operations and does not involve any complex pairing operations, it is faster than BLS signatures. BLS signatures are better to use with our scheme when compared with RSA signatures since BLS signatures are shorter. The size of RSA signatures is equal to the size of RSA modulus. Since the size of RSA modulus we used is 1024 bits, RSA signatures are also of 1024 bits, whereas size of BLS signatures are just 160 bits. In addition, the BLS construction has the shortest query and response. Figure 9 shows the file size to tag set size comparison. Further, in our scheme, the data blocks are of size 128 bytes, and so the tag size will also be 128 bytes, if RSA signatures are used. This will increase the communication cost. RSA signatures are useful if the size of the data blocks is huge. Curtmola et al. [9] uses RSA signatures because they consider data blocks of size 4 KB.

The data update operations are done on multiple file blocks at a time by the data owner. The update operation includes file block insert, modify, delete operations in addition to creation of new file tags and their storage on the cloud. We ran the experiments for file block update on a 1 MB file with three replicas and file block size of 128 bytes. Figure 10a shows the combined User and CSP computation times for file block insert and modify operations, run separately. The experiments are run by inserting and modifying 1–50 % number of file blocks. For example, a file of size



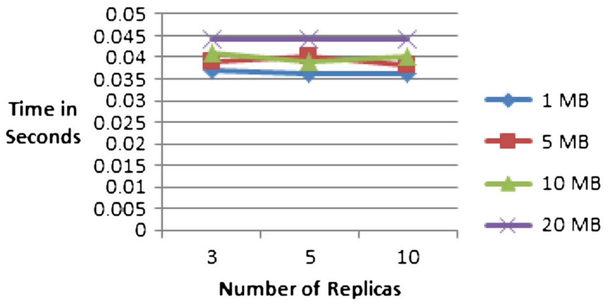
(a) CSP computation time for BLS signatures



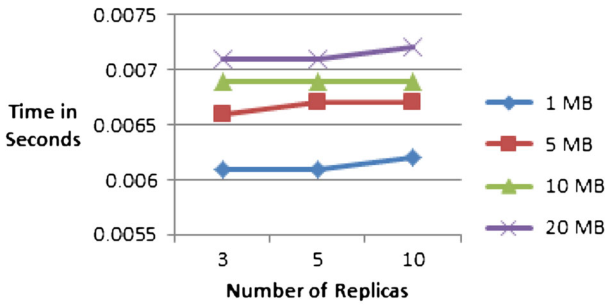
(b) CSP computation time for RSA signatures

Fig. 7 CSP computation time comparison for the number of replicas on the local CentOS server

1 MB has 8192 file blocks. The computation times are calculated by inserting 1 % (≈ 82 file blocks) to 50 % (4096 file blocks) new file blocks and modifying 1 % (≈ 82 file blocks) to 50 % (4096 file blocks) of 8192 file blocks. Figure 10b shows the computation times when the data owner runs both insert and modify operations on a percentage of file blocks. We notice that modify operations take much less time when compared to insert operations. The time taken for modify operation depends on the time taken for paillier multiplication of two 256 byte encrypted file blocks whereas the time taken for insert operation depends on the time taken for writing the 256 byte encrypted file blocks on to the hard drive. We do not calculate the time taken for the file block delete operation since the delete operation does not involve any real User and CSP computations. We also ran our experiments on a micro and large instance in the Amazon EC2 cloud. We used a 64-bit Ubuntu OS with 25GB storage for the micro and large EC2 instances. A micro instance has 613 MB RAM and uses up to 2 EC2 Compute Units whereas a large instance has 7.5GB RAM and uses 4 EC2 Compute Units. Figure 11 shows the comparison of CSP computation times in micro and large instances on the EC2 cloud. We notice that the experiments run a lot faster on the EC2 large instance when compared to the micro instance. EC2 provides instances which have higher configuration than the large instance and the data owner can use them to get better performance. The user computation time is independent of the CSP. We calculated it on the local CentOS 6.3 server which is shown in Fig. 8a. Figures 12



(a) User computation time for BLS signatures

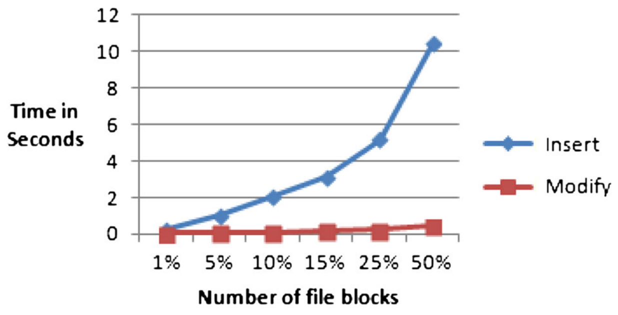


(b) User computation time for RSA signatures

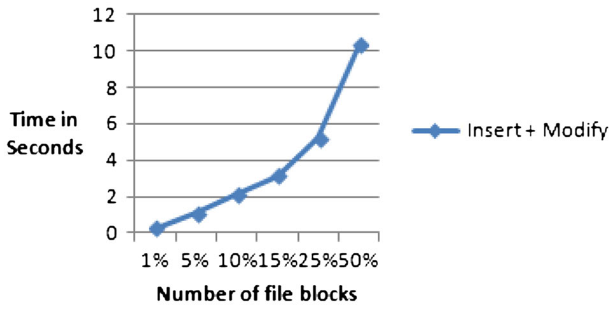
Fig. 8 User computation time comparison for the number of replicas on the local CentOS server

and 13 show the CSP computation times for insert and modify operations on micro and large EC2 instances. It is found that the performance of the update operation is a little faster in large instance when compared to the micro instance. Downloading and uploading file blocks to EC2 micro and large instances take almost the same time. Figure 14 shows the times to download and upload file blocks to EC2 instances.

We also implemented the basic file versioning system ‘MRFVCS’. A file version table is created by the data owner to track the data updates. The number of entries in this table depends on the number of dynamic file block operations performed on the data. The file updates are stored as deltas in the cloud. The delta files generated are of size 128 bytes. To get a particular version of the file, the data owner sends ‘FileVersionRequest’ to the cloud with two parameters ⟨BN, DBN⟩. After receiving ‘FileVersionRequest’, the cloud executes ‘FileVersionDeliver’ algorithm. For a FileVersionRequest with DBN value ‘–’, the file blocks with block number BN are directly delivered to the data owner and does not involve any CSP computation time. For FileVersionRequest with a valid DBN value, the cloud encrypts the file blocks with block number DBN and does a homomorphic addition operation with file blocks with block number BN. The experiments are run on a 1 MB file on local, Amazon EC2 micro and large instances. Figure 15 shows the time taken by the CSP on various instances for executing ‘FileVersionDeliver’ algorithm when a number of FileVersionRequests with valid DBN values is sent by the data owner. We considered FileVersionRequests only with



(a) Time for Insert and Modify operations



(b) Time for Insert + Modify operations

Fig. 9 Time for file block insert and modify operations on the local CentOS server

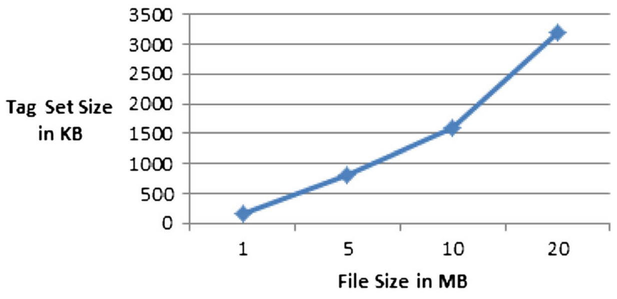
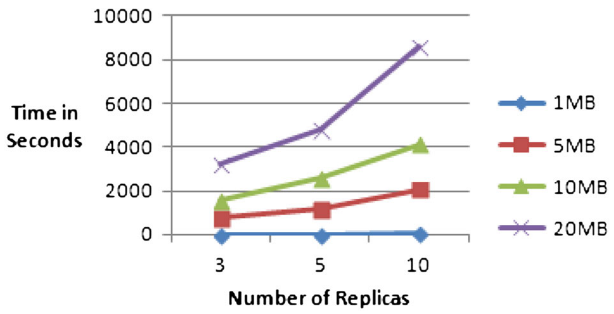


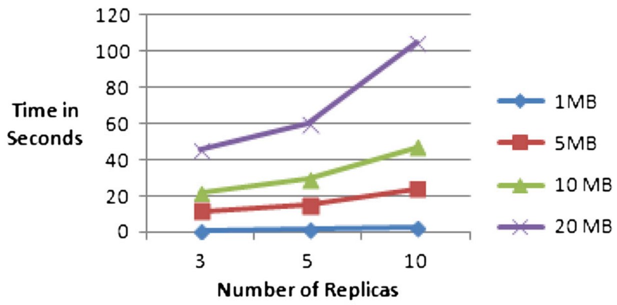
Fig. 10 File size to Tag set size comparison

valid DBN values since there is no computation time involved for delivering file blocks with DBN value '–'.

The number of FileVersionRequests in Fig. 15 is represented in terms of percentage of file blocks. For example, a 1 MB file has 8192 file blocks of size 128 bytes. When the data owner sends 81 FileVersionRequests, the CSP encrypts 81 delta blocks (1 % of 8192 file blocks) and performs 81 homomorphic addition operations. So any number of FileVersionRequests will lead to the CSP performing operations on those number of file blocks and FileVersionRequests can be represented in terms of number of file blocks. MRFVCS does not involve any computation on the part of the data owner



(a) CSP computation time on EC2 micro instance



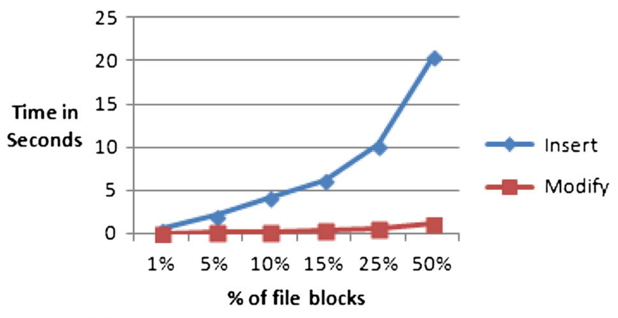
(b) CSP computation time on EC2 large instance

Fig. 11 CSP computation time for number of replicas on Amazon EC2 instances

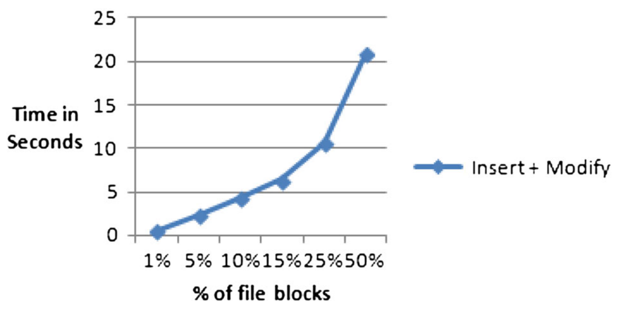
side for executing FileVersionDeliver algorithm and the only cost for the data owner is for maintaining the file version table. The CSP computation time for executing FileVersionDeliver algorithm in MRFVCS is much less compared to the time taken for update operations in the DMR-PDP scheme.

7 Discussion

Figure 6 compares the performance of DMR-PDP scheme with the performance of DMC-PDP proposed in [4]. Figure 6b shows the performance comparison of the CSP computation times. The CSP computation for DMR-PDP and DMC-PDP schemes involve two operations; One on the file tags and the other on the file blocks. The CSP computation time for DMR-PDP scheme is calculated as the time taken by the CSP to perform steps 5, 6 and 7 described in Fig. 3. Barsoum and Hasan [4] describes the procedure to calculate the CSP computation time for DMC-PDP scheme. In DMR-PDP scheme the data owner creates only one set of file tags for all the file replicas, whereas in DMC-PDP scheme the data owner creates file tags for each file replica. So, in DMR-PDP scheme the CSP's operation on the file tags involves less arithmetic operations compared to the number of operations on the file tags performed in DMC-PDP scheme. The CSP's operation on file blocks in both the schemes take almost



(a) Time for Insert and Modify operations



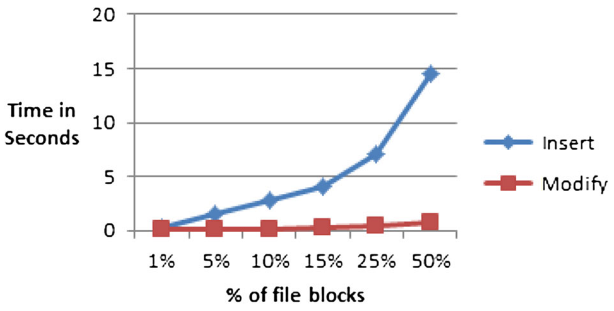
(b) Time for Insert + Modify operations

Fig. 12 Time for file block insert and modify operations in Amazon EC2 micro instance

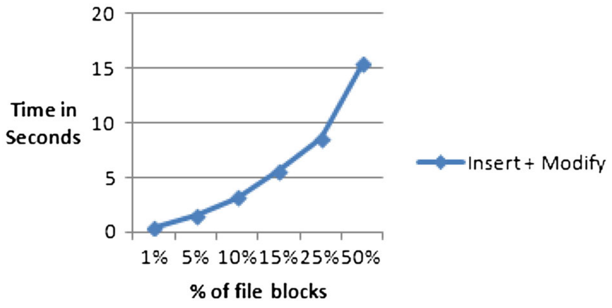
similar time. So, the performance of DMR-PDP scheme in terms of CSP computation time is better than the that of DMC-PDP scheme.

Figure 6c shows the performance comparison of the User computation times for DMR-PDP and DMC-PDP schemes. The User computation time for DMR-PDP scheme is calculated as the time taken by the user to perform steps 8 and 9 described in Fig. 3. Barsoum and Hasan [4] describes the procedure to calculate the User computation time for DMC-PDP scheme. The User computation in DMR-PDP and DMC-PDP schemes are almost similar except for an extra decryption function in DMR-PDP scheme. So, the DMC-PDP scheme performs better in terms of User computation time. But the User computation time for both the schemes differ only in a 1000th of a second. With DMR-PDP schemes, the data owner has the independence to choose the type of file tags. The DMR-PDP scheme works well with BLS signatures and also with RSA signatures. Figures 7 and 8 compares the CSP and User computation times for DMR-PDP scheme when BLS signatures and RSA signatures are used. The experiments showed in Figs. 7 and 8 are performed on the local CentOS server. We ran these experiments on Amazon EC2 micro and large instances. The performance of these experiments are shown in Fig. 11.

Figures 10, 12 and 13 shows the time taken for file block insert, modify and insert + modify operations performed on local CentOS server, Amazon EC2 micro and large instances. The file modify operations are faster than file insert operations due to the



(a) Time for Insert and Modify operations



(b) Time for Insert + Modify operations

Fig. 13 Time for file block insert and modify operations in Amazon EC2 large instance

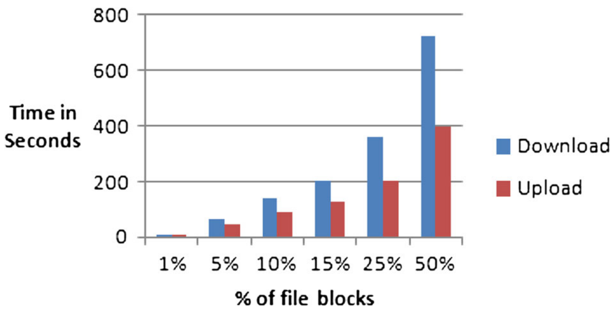


Fig. 14 Download and upload time to EC2 micro and large instances

basic fact that inserting new file blocks involves searching of unused blocks in the hard disk and writing the file block to the disk. We calculated time taken for modify and insert operations by modifying a % of file blocks and also inserting an equal number of new file blocks. Due to higher configuration and on the local CentOS server, these operations perform better when compared to Amazon EC2 large and micro instances. Figure 15 shows the comparision of CSP computation time for FileVersionDeliver algorithm run on local CentOs server, Amazon EC2 micro and large instances. The CSP computation time for FileVersionDeliver algorithm is defined as the time taken by the CSP to deliver the requested version of a % of file blocks to the data owner.

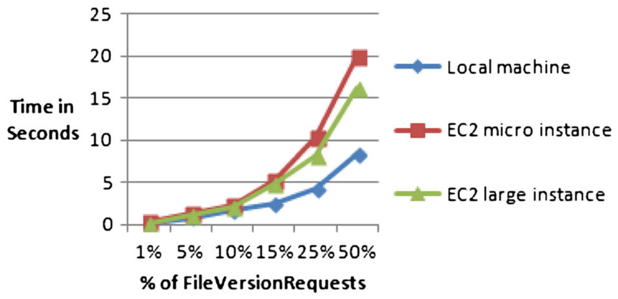


Fig. 15 CSP computation time comparison for FileVersionDeliver algorithm

The FileVersionDeliver algorithm performs better on local CentOS server because of its higher configuration.

8 Conclusion

In this paper, we have presented a scheme for validating the replicated data integrity in a cloud environment. The scheme called DMR-PDP periodically verifies the correctness and completeness of multiple data copies stored in the cloud. Our scheme considers dynamic data update operations on data copies in the verification process. All the data copies can be decrypted using a single decryption key, thus providing a seamless access to all the datas authorized users. The experimental results using the local as well as EC2 cloud instances show that this scheme is better than the previous proposed scheme in terms of dynamic data operations which are performed in much lesser time. In addition, we showed that our scheme works well when extended to support the multiple file versioning where only deltas are stored in the cloud which saves storage cost for the data owner. We believe that these results will help the data owner to negotiate with the cloud provider about the cost and the performance guarantees while maintaining the integrity of the data. Also, these results will provide various incentives to the cloud to take appropriate steps, such as running computations in parallel, to deliver good performance.

References

1. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security, New York, NY, USA, pp. 598–609 (2007).
2. Ateniese, G., Pietro, R.D., Mancin, L.V., Tsudik, G.: Scalable and efficient provable data possession. In SecureComm 08: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, New York, NY, USA, pp. 1–10 (2008).
3. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) Proceedings of SAC. LNCS, vol. 3897, pp. 319–331. Springer-Verlag, Berlin (2005)
4. Barsoum, A.F., Hasan, M.A.: On verifying dynamic multiple data copies over cloud servers. In: Cryptology ePrint Archive, Report 2011/447. http://eprint.iacr.org/ (2011) Please update Refs. [4, 6, 23](16, 21, 13 in original tex file) respectively with last accessed date

5. Boneh, D., Lynn, B., Shacham, H.: 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, pp. 514–532 (2001).
6. Bowers, K.D., van Dijk, M., Juels, A., Oprea, A., Rivest, R.L.: How to tell if your cloud files are vulnerable to drive crashes. In: Cryptology ePrint Archive, Report 2010/214. http://eprint.iacr.org/ (2010)
7. Catalano, D., Gennaro, R., Howgrave-Graham, N., Nguyen, P.Q.: Paillier's Cryptosystem Revisited. ACM CCS '2001. ACM Press, New York (2001).
8. Choi, D.H., Choi, S., Won, D.: Improvement of probabilistic public key cryptosystem using discrete logarithm. In: Kim, K. (ed.) The 4th International Conference on Information Security and Cryptology, ICISC 2001. LNCS, vol. 2288, pp. 72–80. Springer, Berlin (2002)
9. Curtmola, R., Khan, O., Burns, R., Ateniese, G.: MR-PDP: Multiple-Replica Provable Data Possession. In: 28th IEEE ICDCS, pp. 411–420 (2008).
10. Damgard, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public key system. In: 4th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC), pp. 13–15 (2001).
11. Deswarte, Y., Quisquater, J.-J., Sadane, A.: Remote integrity checking. In: Strous, S.J.L. (ed.) 6th Working Conference on Integrity and Internal Control in Information Systems (IICIS), pp. 1–11 (2003).
12. Erway, C., Kupcu, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In CCS 09: Proceedings of the 16th ACM Conference on Computer and Communications Security, New York, NY, USA, pp. 213–222 (2009).
13. Filho, D.L.G., Barreto, P.S.L.M.: Demonstrating data possession and uncheatable data transfer. In: Cryptology ePrint Archive, Report 2006/150 (2006).
14. Golle, P., Jarecki, S., Mironov, I.: Cryptographic primitives enforcing communication and storage complexity. In FC'02: Proceedings of the 6th International Conference on Financial Cryptography, Berlin, Heidelberg, pp. 120–135 (2003).
15. Hao, Z., Zhong, S., Yu, N.: A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability. In: IEEE Transactions on Knowledge and Data Engineering, PrePrints, p. 99 (2011).
16. Mukundan, R., Madria, S., Linderman, M.: Replicated data integrity verification in cloud. In: IEEE Data Engineering Bulletin (2012).
17. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. ACM Trans. Storage **2**(2), 107–138 (2006)
18. Sebe, F., Domingo-Ferrer, J., Martinez-Balleste, A., Deswarte, Y., Quisquater, J.-J.: Efficient remote data possession checking in critical information infrastructures. IEEE Trans. Knowl. Data Eng. **20**, 8 (2008)
19. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) Advances in Cryptology–ASIACRYPT 2008, vol. 5350, pp. 90–107. Springer, Berlin (2008)
20. Shah, M.A., Baker, M., Mogul, J.C., Swaminathan, R.: 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, pp. 1–6 (2007).
21. Shah, M.A., Swaminathan, R., Baker, M.: Privacy-preserving audit and extraction of digital contents. In: Cryptology ePrint Archive, Report 2008/186 (2008).
22. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling public verifiability and data dynamics for storage security in cloud computing. In: ESORICS09: Proceedings of the 14th European Conference on Research in Computer Security, Berlin, Heidelberg, pp. 355–370 (2009).
23. Wang, C., Wang, Q., Ren, K., Lou, W.: Ensuring data storage security in cloud computing. In: Cryptology ePrint Archive, Report 2009/081. http://eprint.iacr.org/ (2009)
24. Zeng, K.: Publicly verifiable remote data integrity. Proceedings of the 10th International Conference on Information and Communications Security, Ser. ICICS '08, pp. 419–434. Springer-Verlag, Berlin (2008)