POR-2P: Network Coding-based POR for Data Provision-Payment System *

Kazumasa Omote and Tran Phuong Thao

Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Nomi, Ishikawa 923-1292
{omote, tpthao}@jaist.ac.jp

Abstract. Proof Of Retrievability (POR) is a protocol that supports a data owner to check whether the data stored in cloud servers is available, intact and retrievable. Based on the POR, network coding technique has been applied to increase efficiency and throughput in data transmission and data repair. Although many network coding-based PORs have been proposed, most of them have not considered a practical scenario in which not only the data owner can check and can retrieve the data stored in the untrusted servers, but also an untrusted user can check and can retrieve the data stored in the servers without learning the secret keys of the data owner. This scenario occurs commonly in reality. For instance, in a *data provision-payment system*, the user must pay money to get data stored in the servers. In this paper, we propose a new network coding-based POR, named POR-2P (a network coding-based POR for data Provision-Payment system), to deal with this scenario. Furthermore, the complexity analysis and the performance evaluation show that the POR-2P is very efficient and applicable for a real cloud system.

Keywords: Cloud Storage, Proof Of Retrievability, Network Coding, Homomorphic MAC

1 Introduction

Since the amount of data is increasing exponentially, storage remote providers called clouds have been proposed to support data owners to reduce the burdens of data storage and data management. However, a cloud provider could be untrusted. Thus, it introduces three data security challenges in data security: availability, integrity and confidentiality. Confidentiality consists of two research approaches: cryptography and information-theoretic. In this paper, we focus on availability, integrity and information-theoretic confidentiality.

Proof Of Retrievability (POR). POR [1–3] has been proposed to support a data owner to check whether his/her data stored in the servers is available, intact and retrievable. POR is a challenge-response protocol between a data owner and a server, and consists of five functions: keygen, encode, check, repair, and retrieve. Based on the POR, the following techniques are commonly used.

- *Replication.* Replication, proposed in [4, 5], is a technique that allows the data owner to store a file replica in each server. The data owner can perform periodic server checks. When a corrupted server is detected, the data owner can use one of the healthy replicas to repair the data in the corrupted server. The drawback of this technique, however, is high storage cost because the data owner must store a whole file in each server.
- *Erasure Coding.* Erasure coding was applied [6, 7] for optimal data redundancy. Instead of storing a file replica in each server like replication, the data owner stores file blocks in each

* This study is partly supported by Grant-in-Aid for Grant-in-Aid for Young Scientists (B) (25730083) and CREST, JST.

server. Thus, the storage cost can be reduced. However, the drawback of this technique is that: in order to repair a corrupted server, the data owner must reconstruct the original file before repairing. Hence, the computation cost is increased during data repair.

- *Obvious RAM (ORAM).* Recently, the ORAM has been applied to the POR [8,9]. Basically, this technique was proposed for privacy-preserving data access pattern. By using ORAM structure, the servers cannot obtain the access patterns when the data owner performs data checks. For data repair, ORAM-based POR embeds the erasure coding to repair the corruption. The ORAM structure leads to high storage cost because of its hierarchical storage layout. The ORAM structure also leads to high computation cost because of its shuffling procedure every a number of read/write operations.
- *Network Coding.* To address the drawback of erasure coding, network coding has been applied [10–12] to improve efficiency and throughput in data transmission and data repair. Unlike erasure coding, the data owner does not need to reconstruct the entire file before generating new coded blocks. Instead, the coded blocks which are collected from healthy servers can be used to generate new coded blocks. Compare with the ORAM, the structure of network coding is much simpler. It has no hierarchical storage, no shuffling procedure like ORAM, and no the drawback of the erasure coding. Therefore, this paper focuses on network coding.

MAC vs. signature. The data stored in the servers cannot be checked without additional authentication information, i.e., Message Authentication Code (MAC) or signature. A MAC is also called a *tag*. The traditional MAC and signature are not suitable for network coding. Thus, new techniques called homomorphic MAC [14–16] and homomorphic signature [17, 18] have been proposed. Furthermore, a MAC is used in a symmetric key setting while a signature is used in an asymmetric key setting. Because this paper focuses on a symmetric key setting for efficiency, we thus use homomorphic MAC in our scheme.

Contribution. In this paper, we propose a network coding-based POR (named POR-2P) which has the following contributions:

1. POR-2P can deal with *data provision-payment system* in which not only data owner and servers but also user can participate in the system. The user pays money to get data stored in the servers. This system exists two security problems:
   - *The user may not pay enough money*: POR-2P can address this problem by only allowing the user to retrieve a partial data (called data piece), not whole data each time. After getting a data piece, the user must pay money before getting the next data piece. The user can retrieve the whole data if he gathers enough data pieces. In other words, if the number of retrieved data pieces is larger than or equal to a threshold, the whole data can be fully recovered. That is the reason why we choose information-theoretic confidentiality in this paper.
   - *The servers may not provide the valid data*: POR-2P can address this problem by requiring the user to check each data piece provided from the servers. Herein introduces the following challenge. The data is encoded by the data owner before being stored in the untrusted servers. Furthermore, the user is also untrusted. Thus, the data owner cannot give his/her secret keys, which are used to compute the MACs, to the user. Therefore, how can the user check whether the data piece provided from the servers is valid without having any information about the secret keys of the data owner? POR-2P can address this challenge by using a technique called orthogonal keygen.
2. POR-2P is constructed based on a symmetric key setting which is well-known to be more efficient than asymmetric key setting. The user can check the data piece sent by the untrusted servers without any public key.
3. POR-2P can deal with access control during retrieve process by using our key management. Concretely, data owner can control which data that the user is allowed to retrieve and which data that the user is not allowed to retrieve.

4. The complexity analysis of POR-2P show that its communication and computation costs are lower than these costs in some comparable previous schemes. The performance evaluation shows that POR-2P is very applicable to a real system.

Roadmap. The related work is introduced in Section 2. The preliminaries are described in Section 3. The system model and adversarial model are presented in Section 4 and 5. The POR-2P scheme is proposed in Section 6. The security and efficiency analyses, and the performance evaluation are discussed in Section 7 and 8. Finally, the conclusion and future work are drawn in Section 9.

2 Related Work

There are a few notable network coding-based PORs. Dimakis et al. [12] are the first to apply network coding for distributed storage systems. Their work achieved a remarkable reduction in the communication overhead of the repair component. Acedanski et al. [13] demonstrated that when the random linear coding is applied to distributed storage system, it performs as well without suffering additional storage space required at the centralized server before distribution among multiple locations. Further, with a probability close to one, the minimum number of storage location a downloader needs to connect to (for reconstructing the entire file), can be very close to the case where there is complete coordination between the storage locations and the downloader. Li et al. [19] proposed a tree-structure data regeneration with the linear network coding to achieve an efficient regeneration traffic and bandwidth capacity by using an undirected-weighted maximum spanning tree and the Prim algorithm. Chen et al. then proposed RDC-NC [20] (Remote Data Checking for Network Coding-based distributed storage systems) which provides a decent solution for efficient data repair by recoding encoded blocks in the healthy servers. Chen et al. proposed NC-Cloud [21] (a Network-Coding-based storage system in a Cloud-of-Clouds) to improve cost-effective repair using the functional minimum-storage regenerating (FMSR) code and to relax the encoding requirement of storage nodes during repair. NC-Audit [22] (Auditing for Network Coding storage) was proposed also for efficient data check and repair and data leakage using a combination of a homomorphic MAC (called SpaceMac) and a chosen-plaintext attack (CPA)-secure encryption (called NCrypt).

In most of these previous schemes, the system model consists of two entities: (i) a data owner who outsources, encodes and checks his/her data, and (ii) servers which store the data and provide proofs to the data owner. Some papers deal with multiple data owners [23, 24]. However, none of previous schemes considers the very practical scenario that: besides data owner(s) and servers, the system model should consist of one more entity which is user. This user can retrieve the data stored in the servers. For instance, in a *data provision-payment system*, the user pays money to get data stored in the servers.

3 Preliminaries

3.1 Proof Of Retrievability (POR)

The POR [1–3], which is a challenge-response protocol between a data owner (verifier) and a server (prover), was proposed to help the verifier to check whether his/her data stored in the server is available, intact and retrievable. The POR has the following functions:

- $\mathsf{keygen}(1^\lambda)$: The data owner executes this function which inputs a security parameter (λ) and outputs a secret key (sk) along with a public key (pk). For a symmetric key system, pk is set to be null.
- $\mathsf{encode}(\mathsf{sk}, F)$: This function allows the data owner to encode a raw file (F) to an encoded file (F'). F' is then stored in the server.

- $\mathsf{check}(\mathsf{sk})$: This function conducts the challenge-response protocol between the data owner and the server during which the data owner uses sk to generate a challenge (c) and sends the c to the server. The server computes a corresponding response (r) and sends the r back to the data owner. The data owner then verifies the server based on c and r.
- $\mathsf{retrieve}(c_0, \cdots, c_{m-1})$: This function is performed when the data owner wants to retrieve F based on a set of coded blocks of some servers. This function selects some healthy servers and requests those selected servers to send their coded blocks (c).
- $\mathsf{repair}()$: When a corrupted data from a server is detected in the check function, this function is executed by the data owner to repair the corrupted data. The repair function is depended on the used techniques, e.g., replication, erasure coding, or network coding.

3.2 Network Coding

The network coding [10–12] was proposed for cost-efficiency in data transmission. Suppose that a data owner owns a file F and wants to store the redundant coded blocks in the servers in a way that the data owner can reconstruct F and can repair the coded blocks in a corrupted server. The data owner firstly divides F into m blocks: $F = v_1||\cdots||v_m \in \mathbb{F}_q^z$. $v_k \in \mathbb{F}_q^z$ where $k \in \{1, \cdots, m\}$ and $\mathbb{F}_q^z$ denotes a z-dimensional finite field of a prime order q. The data owner then augments v_k with a vector of length m which contains a single '1' in the position k and $(m-1)$ single '0's elsewhere. The resulting block is called *augmented block* (says, w_k). w_k has the following form:

$$w_k = (v_k, \overbrace{\underbrace{0, \cdots, 0, 1}_{k}, 0, \cdots, 0}^{m}) \in \mathbb{F}_q^{z+m} \tag{1}$$

Thereafter, the data owner randomly chooses m coefficients $\alpha_1, \cdots, \alpha_m \in \mathbb{F}_q$ to compute coded blocks using the linear combination:

$$c = \sum_{k=1}^{m} \alpha_k \cdot w_k \in \mathbb{F}_q^{z+m} \tag{2}$$

The data owner stores these coded blocks in the servers. To reconstruct F, any m coded blocks are required to solve m augmented blocks $w_1, \cdots, w_m$ using the accumulated coefficients contained in the last m coordinates of each coded block. After the m augmented blocks are solved, m file blocks $v_1, \cdots, v_m$ are obtained from the first coordinate of each augmented block. Finally, F is reconstructed by concatenating the file blocks. Note that the matrix consisting of the coefficients used to construct any m coded blocks should have full rank. Koetter et al. [25] proved that if the prime q is chosen large enough and the coefficients are chosen randomly, the probability for the matrix to have full rank is high. When a server is corrupted, the data owner repairs it by retrieving the coded blocks from the healthy servers and linearly combining them to regenerate the new coded blocks. An example of the data repair is given in Figure 1.

3.3 Homomorphic MAC

Inner-product MAC. This MAC is the simplest homomorphic MAC which consists of the following algorithms:

- $\mathsf{Gen}(1^\lambda) \to k$: the algorithm inputs a security parameter λ and outputs a secret key k.
- $\mathsf{Tag}_k(M) \to t$: the algorithm inputs k and a message M; and outputs a tag t such that:

$$t = M \cdot k \tag{3}$$

- $\mathsf{Ver}_k(M, t) \to \{0, 1\}$: the algorithm inputs M, t and k; and outputs 1 if t is a valid tag. Otherwise, it outputs 0.

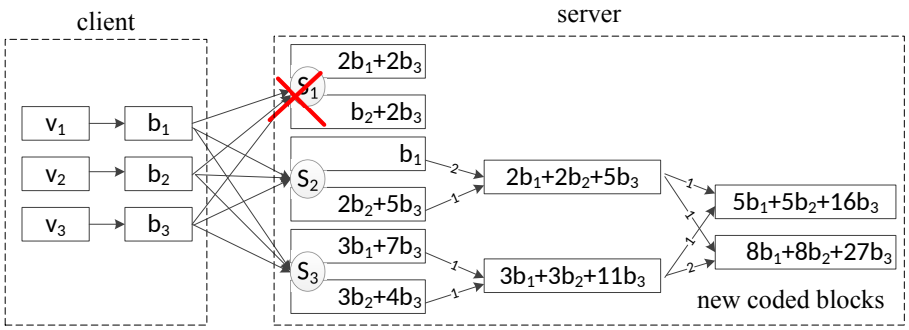

Fig. 1: An example of data repair in network coding

Wegman-Carten MAC. The inner-product MAC nay not be secured from the response replay attack (see Section 5.2) when it is combined with the network coding. The Wegman-Carten MAC was then introduced as follows:

- $\mathsf{Gen}(1^\lambda) \rightarrow \{k, k'\}$: the algorithm inputs a security parameter λ and outputs secret keys $\{k, k'\}$. k is used for tagging the message M. k' is used for permuting the tag.
- $\mathsf{Tag}_{\{k,k'\}}(M) \rightarrow t$: the algorithm inputs $\{k, k'\}$ and a message M; and outputs a tag t such that:

$$t = M \cdot k + f_{k'}(r) \tag{4}$$

  where f and r denote a pseudorandom function and a randomness, respectively.
- $\mathsf{Ver}_{\{k,k'\}}(M, t) \rightarrow \{0, 1\}$: outputs 1 if t is a valid tag, and 0 otherwise.

This Wegman-Carter MAC will be used in our scheme.

3.4 Orthogonal Keygen

The technique is proposed in [26] to to generate a key such that it is orthogonal to all the augmented blocks. In a formal statement, given the set of augmented blocks $\{w_1, \cdots, w_m\}$, the algorithm outputs a key k such that $k \cdot w_i = 0$ for all $i = 1, \cdots, m$. The algorithm is given as follows.

$\mathsf{OrthogonalKeygen}$ $(w_1, \cdots, w_m) \rightarrow k$: Let π denote the span of $w_1, \cdots, w_m \in \mathbb{F}_q^{z+m}$. Let M be the matrix in which each of the m augmented blocks is a row of M. $\mathsf{rank}(M) = m$. Let π_M be the space spanned by the rows of M. The null space of M, denoted by $\pi_M^\perp$, is the set of all vectors $u \in \mathbb{F}_q^{z+m}$ such that $M \cdot u^T = 0$. For any $m \times (z+m)$ matrix, the rank-nullity theorem gives:

$$\mathsf{rank}(M) + \mathsf{nullity}(M) = z + m \tag{5}$$

where $\mathsf{nullity}(M)$ is the dimension of $\pi_M^\perp$. And thus,

$$\mathsf{dim}(\pi_M^\perp) = (z + m) - m = z \tag{6}$$

Let $\{b_1, \cdots, b_z\} \in \mathbb{F}_q^{z(z+m)}$ be a basis of $\pi_M^\perp$ which can be found by solving $M \cdot u^T = 0$. Let f be a Pseudorandom function such that $f : \mathcal{K} \times [1, z] \rightarrow \mathbb{F}_q$. The key k is computed as:

- $r_i \leftarrow f(k_{PRF}, i) \in \mathbb{F}_q$ for $i \in \{1, \cdots, z\}$ and $k_{PRF} \in \mathcal{K}$.
- $k \leftarrow \sum_{i=1}^{z} r_i \cdot b_i \in \mathbb{F}_q^{z+m}$.

4 System Model

4.1 System Entities

The system model consists of the following three types of entities:

– *Data owner*: Data owner is a trusted entity who owns a data and wants to store the data in the servers. Before storing the data in the servers, the data owner encodes the data and uses his/her secret keys to compute tags for the data. The data owner will check the servers periodically. If a corrupted server is detected, he will repair the corruption using network coding.
– *User*: User is an untrusted entity who wants to retrieve the data of the data owner. The user is given keys by the data owner (not the secret keys of the data owner) to check the response from the servers during data retrieval.
– *Servers*: Servers are the untrusted entities who store the data. The servers have responsibility to provide the proofs along with a partial data to the data owner (during the check and repair procedures) and to the user (during retrieve procedure).

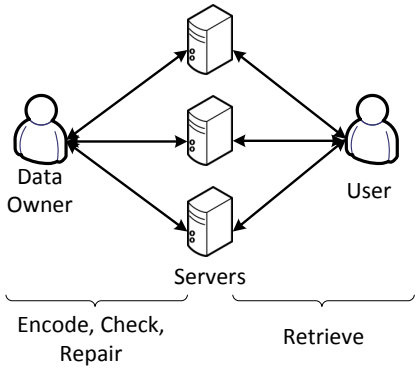

Fig. 2: The system model

4.2 System Requirements and Assumptions

– Only the data owner can compute the tags of the data using his/her secret keys. Although the user is given keys (which is different from the secret keys of the data owner) to check and to repair the servers, the user cannot compute the tags for the data.
– The user and the servers do not collude with each other.
– All the keys are transmitted via a secure channel.

5 Adversarial Model

5.1 Attacks From User

The user performs the following two attacks:

– The user searches the secret keys of the data owner via two ways: (i) brute force search from the key of the user, (ii) orthogonal keygen after retrieving the data.
– The user forges a tag for a *new* file block. Note that the user only forges a tag for a new file block because for an old file block, the tag is already computed by the data owner. Thus, forging a tag for an old file block is meaningless. The security from tag forgery of a new file block will be proved in Theorem 3 in Section 7.

5.2 Attacks From Servers

Response Replay Attack. This attack means that a malicious server reuses the correct response in the past check phase in order to save the computation cost, and is still able to pass the verification in the current check phase. For example:

- Epoch 1: the data owner sends a challenge Q to the server $S_{\mathcal{A}}$. $S_{\mathcal{A}}$ responds $\{c_{\mathcal{A}}, t_{\mathcal{A}}\}$ where $c_{\mathcal{A}}$ and $t_{\mathcal{A}}$ denote the aggregated coded block and aggregated tag, respectively. The data owner verifies $S_{\mathcal{A}}$ by checking whether $t_{\mathcal{A}} = c_{\mathcal{A}} \cdot k_C$ where k_C denotes the secret key of the data owner. Suppose that the equality holds.
- Epoch 2: the data owner sends a challenge Q' to $S_{\mathcal{A}}$. $S_{\mathcal{A}}$ reuses $\{c_{\mathcal{A}}, t_{\mathcal{A}}\}$ which are generated in the epoch 1 without computing another response for Q'. $S_{\mathcal{A}}$ still passes the verification $t_{\mathcal{A}} = c_{\mathcal{A}} \cdot k_C$.

Pollution Attack. This attack means that a malicious server responds a valid coded block to pass the check phase, but then provides an invalid coded block in the repair phase in order to prevent data recovery. For example:

- Encode: the data owner encodes the augmented blocks $\{w_1, w_2, w_3\}$ into six coded blocks:
  - $c_{11} = w_1$ and $c_{12} = w_2 + w_3$.
  - $c_{21} = w_3$ and $c_{22} = w_1 + w_2$.
  - $c_{31} = w_1 + w_3$ and $c_{32} = w_2 + w_3$.

  Then, $\{c_{11}, c_{12}\}$ are stored in the server S_1; $\{c_{21}, c_{22}\}$ are stored in the server S_2; and $\{c_{31}, c_{32}\}$ are stored in the server S_3.
- Check: the corrupted S_3 is detected.
- Repair: to repair S_3, the data owner requires S_1 and S_2 to provide their aggregated coded blocks. S_1 provides $c_1 = 1c_{11} + 1c_{12} = w_1 + w_2 + w_3$. Suppose S_2 injects a polluted block $c_2 = X$ which is an arbitrary value. The data owner computes two new coded blocks for the new server S'_3: $c'_{31} = 1c_1 + 1c_2 = X + w_1 + w_2 + w_3$, and $c'_{32} = 1c_1 + 2c_2 = 2X + w_1 + w_2 + w_3$.

The number of unknowns is now $(m + 1)$ where m denotes the number of augmented blocks (in this example, $m = 4$ and the unknowns are X, w_1, w_2 and w_3). Therefore, the original file cannot be retrieved from any m or more coded blocks.

6 Our Proposed POR-2P Scheme

The notations used throughout our scheme are given as follows:

- $\mathcal{DO}$ denotes the data owner.
- $\mathcal{U}$ denotes the user.
- n denotes the number of servers.
- $\mathcal{S}_i$ denotes a server where $i \in \{1, \cdots, n\}$.
- F denotes the original file.
- m denotes the number of file blocks.
- $\mathbb{F}_q^z$ denotes the z-dimensional finite field of a prime order q.
- v_k denotes a file block where $k \in \{1, \cdots, m\}$.
- w_k denotes an augmented block of v_k where $k \in \{1, \cdots, m\}$.
- d denotes the number of coded blocks stored in a server.
- c_{ij} denotes a coded block where $i \in \{1, \cdots, n\}, j \in \{1, \cdots, d\}$.
- t_{ij} denotes the tag of c_{ij} where $i \in \{1, \cdots, n\}, j \in \{1, \cdots, d\}$.
- s denotes the number of spot checks where $1 \leq s \leq d$.
- $\mathcal{S}_r$ denotes a corrupted server.
- $\mathcal{S}'_r$ denotes a new server used to replace $\mathcal{S}_r$.
- f denotes a pseudo-random function: $\{0,1\}^* \times \{0,1\}^\kappa \rightarrow \mathbb{F}_q$.

6.1 Setup

1. *Augmented blocks*: $\mathcal{DO}$ divides F into m file blocks $F = v_1 || \cdots || v_m$. $v_k \in \mathbb{F}_q^z$ where $k \in \{1, \cdots, m\}$. $\mathcal{DO}$ creates m augmented blocks. Each augmented block has the following form:

$$w_k = (v_k, \overbrace{\underbrace{0, \cdots, 0, 1}_{k}, 0, \cdots, 0}^{m}) \in \mathbb{F}_q^{z+m} \tag{7}$$

2. *Keygen:* $\mathcal{DO}$ generates the following keys:
   - $k_{DO} \xleftarrow{rand} \mathbb{F}_q^{z+m}$.
   - $k_\phi \in \mathbb{F}_q^{z+m} \leftarrow \mathsf{OrthogonalKeygen}(w_1, \cdots, w_m)$. The property of k_ϕ is that it is orthogonal to each of $\{w_1, \cdots, w_m\}$. Formally, $k_\phi w_k = 0$ for all $k \in \{1, \cdots, m\}$.
   - $k_{PRF} \xleftarrow{rand} \mathbb{F}_q^\kappa$.

   $\mathcal{DO}$ then computes:
$$k_U = k_{DO} + k_\phi \in \mathbb{F}_q^{z+m} \tag{8}$$
   $\mathcal{DO}$ sends $\{k_U, k_{PRF}\}$ to $\mathcal{U}$. $\mathcal{DO}$ keeps $\{k_{DO}, k_\phi, k_{PRF}\}$.

6.2 Encode

1. $\mathcal{DO}$ computes nd coded blocks and nd tags as follows:
   For $\forall i \in \{1, \cdots, n\}, \forall j \in \{1, \cdots, d\}$:
   - $\mathcal{DO}$ generates m coefficients: $\alpha_{ijk} \xleftarrow{rand} \mathbb{F}_q$.
   - $\mathcal{DO}$ computes coded block:
$$c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} \cdot w_k \in \mathbb{F}_q^{z+m} \tag{9}$$
   - $\mathcal{DO}$ computes tag:
$$t_{ij} = k_{DO} \cdot c_{ij} + f_{k_{PRF}}(i||j) \in \mathbb{F}_q \tag{10}$$
2. For $\forall j \in \{1, d\}$, $\mathcal{DO}$ sends $\{c_{ij}, t_{ij}\}$ to $\mathcal{S}_i$.

6.3 Check (spot check)

1. $\mathcal{DO}$ generates the challenge Q as follows:
   - $\mathcal{DO}$ generates $s \xleftarrow{rand} \{1, \cdots, d\}$. s can be different for each server and each check time.
   - $\mathcal{DO}$ generates the challenge Q consisting of s pairs $(b_1, \beta_1), \cdots, (b_s, \beta_s)$ where $b_u \xleftarrow{rand} \{1, \cdots, d\}$ and $\beta_u \xleftarrow{rand} \mathbb{F}_q$ for $u \in \{1, \cdots, s\}$.
   - $\mathcal{DO}$ sends $Q = \{(b_1, \beta_1), \cdots, (b_s, \beta_s)\}$ to $\mathcal{S}_i$.
2. $\mathcal{S}_i$ responds $\mathcal{DO}$ as follows:
   - $\mathcal{S}_i$ combines coded blocks:
$$c_i = \sum_{u=1}^{s} \beta_u \cdot c_{ib_u} \in \mathbb{F}_q^{z+m} \tag{11}$$
   - $\mathcal{S}_i$ combines tags:
$$t_i = \sum_{u=1}^{s} \beta_u \cdot t_{ib_u} \in \mathbb{F}_q \tag{12}$$
   - $\mathcal{S}_i$ sends $\{c_i, t_i\}$ to $\mathcal{DO}$.
3. $\mathcal{DO}$ verifies $\mathcal{S}_i$ as follows:
   - $\mathcal{DO}$ computes:
$$t'_i = k_{DO} c_i + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u) \tag{13}$$
   - $\mathcal{DO}$ checks whether the following equation holds. If it holds, S_i is healthy. Otherwise, S_i is corrupted.
$$t'_i = t_i. \tag{14}$$

The correctness of Eq. 14 is proved as follows:

Proof.
$t'_i = k_{DO} c_i + \sum_{u=1}^{s} \beta_u f_{k_{PRF}}(i||b_u)$ //because of Eq. 13
$= \sum_{u=1}^{s} k_{DO} \beta_u c_{ib_u} + \sum_{u=1}^{s} \beta_u f_{k_{PRF}}(i||b_u)$ // because of Eq. 11
$= \sum_{u=1}^{s} \beta_u (k_{DO} c_{ib_u} + f_{k_{PRF}}(i||b_u))$
$= \sum_{u=1}^{s} \beta_u t_{ib_u}$ //because of Eq. 10 with $j = b_u$
$= t_i$ //because of Eq. 12

6.4 Repair

When a corrupted server ($\mathcal{S}_r$) is detected, the new server ($\mathcal{S}'_r$) is used to replace $\mathcal{S}_r$.

1. $\mathcal{DO}$ performs as in the check phase until he gets enough d valid aggregated coded blocks (which are contained in the server responses), says, $R_c = \{c_{i_1}, \cdots, c_{i_d}\}$.
2. $\mathcal{DO}$ computes d coded blocks and d tags for S'_r as follows:
   - $\mathcal{DO}$ assigns d valid aggregated coded blocks as the new coded blocks of $\mathcal{S}'_r$: $(c_{r_1}, \cdots, c_{r_d}) \leftarrow (c_{i_1}, \cdots, c_{i_d})$.
   - $\mathcal{DO}$ computes new tags: $t_{rj} = c_{rj} \cdot k_U + f_{k_{PRF}}(r||j)$ where $j \in \{1, \cdots, d\}$.
3. $\mathcal{DO}$ sends $\{c_{rj}, t_{rj}\}$ to S'_r where $j \in \{1, \cdots, d\}$.

6.5 Retrieve

When $\mathcal{U}$ wants to retrieve F, firstly, he performs similarly to the check phase in order to collect enough m coded blocks. $\mathcal{U}$ then solves m file blocks and recovers F by concatenating all the file blocks. The difference between the check phase and the retrieve phase is that $\mathcal{DO}$ checks the servers using k_{DO} and k_{PRF} while in the retrieve phase, $\mathcal{U}$ checks the servers using k_U and k_{PRF}. Concretely, $\mathcal{U}$ performs as follows:

1. $\mathcal{U}$ generates the challenge Q as follows:
   - $\mathcal{U}$ generates $s \overset{rand}{\leftarrow} \{1, \cdots, d\}$.
   - $\mathcal{U}$ generates the challenge Q consisting of s pairs $(b_1, \beta_1), \cdots, (b_s, \beta_s)$ where $b_u \overset{rand}{\leftarrow} \{1, \cdots, d\}$ and $\beta_u \overset{rand}{\leftarrow} \mathbb{F}_q$ for $u \in \{1, \cdots, s\}$.
   - $\mathcal{U}$ sends $Q = \{(b_1, \beta_1), \cdots, (b_s, \beta_s)\}$ to $\mathcal{S}_i$.
2. $\mathcal{S}_i$ responds $\mathcal{U}$ as follows:
   - $\mathcal{S}_i$ combines coded blocks:

$$c_i = \sum_{u=1}^{s} \beta_u \cdot c_{ib_u} \in \mathbb{F}_q^{z+m} \tag{15}$$

   - $\mathcal{S}_i$ combines tags:

$$t_i = \sum_{u=1}^{s} \beta_u \cdot t_{ib_u} \in \mathbb{F}_q \tag{16}$$

   - $\mathcal{S}_i$ sends $\{c_i, t_i\}$ to $\mathcal{U}$.
3. $\mathcal{U}$ verifies $\mathcal{S}_i$ as follows:
   - $\mathcal{U}$ computes:

$$t'_i = k_U c_i + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u) \tag{17}$$

   - $\mathcal{U}$ checks whether the following equation holds. If it holds, S_i is healthy; otherwise, S_i is corrupted.

$$t'_i = t_i. \tag{18}$$

The correctness of Eq. 18 is proved as follows:

Proof.
$t'_i = k_U c_i + \sum_{u=1}^{s} \beta_u f_{k_{PRF}}(i||b_u)$ //because of Eq. 17
$= (k_{DO} + k_\phi) c_i + \sum_{u=1}^{s} \beta_u f_{k_{PRF}}(i||b_u)$ //because of Eq. 8
$= (k_{DO} + k_\phi) \sum_{u=1}^{s} \beta_u c_{ib_u} + \sum_{u=1}^{s} \beta_u f_{k_{PRF}}(i||b_u)$ //because of Eq. 15
$= (k_{DO} + k_\phi) \sum_{u=1}^{s} \sum_{k=1}^{m} \beta_u \alpha_{ib_u k} w_k + \sum_{u=1}^{s} \beta_u f_{k_{PRF}}(i||b_u)$
//because of Eq. 9 when replacing $j = b_u$
$= k_{DO} \sum_{u=1}^{s} \sum_{k=1}^{m} \beta_u \alpha_{ib_u k} w_k + \sum_{u=1}^{s} \beta_u f_{k_{PRF}}(i||b_u)$
// because $k_\phi w_k = 0, \forall k \in \{1, \cdots, m\}$
$= k_{DO} \sum_{u=1}^{s} \beta_u c_{ib_u} + \sum_{u=1}^{s} \beta_u f_{k_{PRF}}(i||b_u)$ // because of Eq. 9 with $j = b_u$
$= \sum_{u=1}^{s} \beta_u (k_{DO} c_{ib_u} + f_{k_{PRF}}(i||b_u))$
$= \sum_{u=1}^{s} \beta_u t_{ib_u}$ // because of Eq. 10 with $b_u = j$
$= t_i$ // because of Eq. 16
□

4. After collecting enough m aggregated coded blocks, $\mathcal{U}$ computes m augmented blocks $\{w_1, \cdots, w_m\}$ by using Gaussian elimination, then obtains m file blocks $\{v_1, \cdots, v_m\}$ which are contained in the first element of each augmented block. Finally, $\mathcal{U}$ recovers the original file as $F = v_1||\cdots||v_m$.

Access control. In this part, we show that how POR-2P can address access control in retrieval process. Consider the scenario that $\mathcal{DO}$ owns multiple files and there are multiple users. Each user has a different access privilege to each file. To deal with this challenge, our main idea is that if $\mathcal{DO}$ wants a user to retrieve which files, $\mathcal{DO}$ will generate the key k_ϕ such that it is orthogonal to those files.

Concretely, suppose that $\mathcal{DO}$ owns h files: $\{F_1, \cdots, F_h\}$. Let x denote file index. Then, for each $x \in \{1, \cdots, g\}$, $\mathcal{DO}$ performs:

- Divide F_x into m blocks: $F_x = v_{x_1}||\cdots||v_{xm}$.
- Create augmented blocks: $\{w_{x1}, \cdots, w_{xm}\}$.
- Compute coded blocks: $\{c_{xi1}, \cdots, c_{xid}\}$ for each $i \in \{1, \cdots, n\}$.
- Compute tags: $\{t_{xi1}, \cdots, t_{xid}\}$ for each $i \in \{1, \cdots, n\}$.

Suppose that there are h users $\{\mathcal{U}_1, \cdots, \mathcal{U}_h\}$ participating in the system. Suppose that $\mathcal{DO}$ allows $\mathcal{U}_y$ where $y \in \{1, \cdots, h\}$ to retrieve a subset of files: $\Gamma_y = \{F_{x_1}, \cdots, F_{x_r}\} \subseteq \{F_1, \cdots, F_g\}$. To enable this access control, $\mathcal{DO}$ manages the keys as follows:

- $k_{DO} \xleftarrow{rand} \mathbb{F}_q^{z+m}$.
- $k_{\phi_y} \in \mathbb{F}_q^{z+m} \leftarrow \mathsf{OrthogonalKeygen}(w_{x1}, \cdots, w_{xm})$ for all $x \in \{x_1, \cdots, x_r\}$. The property of k_{ϕ_y} is that it is orthogonal to all augmented blocks of the files which belong to Γ_y. Formally, $k_{\phi_y} w_{x1} = \cdots = k_{\phi_y} w_{xm} = 0$ for all $x \in \{x_1, \cdots, x_r\}$.
- $k_{PRF} \xleftarrow{rand} \mathbb{F}_q^{\kappa}$.

Then, $\mathcal{DO}$ sends $k_{U_y} = k_{DO} + k_{\phi_y}$ to $\mathcal{U}_y$. *Note that k_{DO} and k_{PRF} are the same for all users, but only k_{ϕ_y} is different for each user according to which files that the user is allowed to retrieve.*

7 Security Analysis

Before describing how the POR is secured from the attacks in the adversarial model, we give the data retrieval condition as follows.

Theorem 1. *F can be retrieved if in any epoch, at least m coded blocks are healthy.*

Proof. $F = v_1||\cdots||v_m$. Thus, to retrieve F, we view $v_1, \cdots, v_m$ as m unknowns that need to be solved. Because each coded block is computed from a linear combination of all m file blocks, to solve these m coded block, we need at least m coded blocks to make the linear coefficient matrix have full rank. □

7.1 Attacks From User

We consider the probability for $\mathcal{U}$ to search for the secret keys $\{k_{DO}, k_\phi\}$ of $\mathcal{DO}$.

Theorem 2. *Given k_U, $\{k_{DO}, k_\phi\}$ cannot be derived via the brute force search or orthogonal keygen.*

Proof. Because $k_U = k_{DO} + k_\phi \in \mathbb{F}_q^{z+m}$, the probability to find k_{DO} and k_ϕ via brute force search is $1/q^{z+m}$. If q is chosen large enough (i.e., 160 bits), the probability to find $\{k_{DO}, k_\phi\}$ is $1/(2^{160})^{(z+m)}$, which is negligible.

Besides the brute force search, $\mathcal{U}$ may search $\{k_{DO}, k_\phi\}$ as follows: after getting enough m coded blocks and retrieving all m augmented blocks, $\mathcal{U}$ runs the orthogonal keygen to find all basis vectors which are orthogonal to all m augmented blocks to obtain k_ϕ. Let $\{b_1, \cdots, b_z\}$

be the basis vectors found by $\mathcal{U}$ (as mentioned in the orthogonal keygen). k_ϕ is computed as: $k_\phi \leftarrow \sum_{i=1}^{z} r_i \cdot b_i \in \mathbb{F}_q^{z+m}$ where $r_i \in \mathbb{F}_q$ is randomly generated by a pseudo random function. Therefore, the probability for $\mathcal{U}$ to obtain k_ϕ is z/q. This means that $\mathcal{U}$ has to guess all z random r_i's to obtain k_ϕ. If q is chosen large enough (i.e., 160 bits), the probability to find k_ϕ is $z/2^{160}$, which is negligible. □

We now prove that $\mathcal{U}$ cannot compute the tag for any new file block. For the old file blocks, $\mathcal{U}$ does not have any purpose to compute the tags because the tags already exist (computed by $\mathcal{DO}$).

Theorem 3. *Given a new file block v_π ($v_\pi \neq v_1, \cdots, v_m$) and given the key $k_U = k_{DO} + k_\phi$, $\mathcal{U}$ cannot forge the tag for v_π.*

Proof. Given v_π, the new coded block is computed as:

$$c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} w_k + \alpha_\pi w_\pi \in \mathbb{F}_q^{z+m} \tag{19}$$

where $\alpha_\pi \overset{rand}{\leftarrow} \mathbb{F}_q$. Suppose that $\mathcal{U}$ tries to compute the tag for c_{ij} using k_U and k_{PRF} as follows:

$t_{ij} = k_U \cdot c_{ij} + f_{k_{PRF}}(i||j) \in \mathbb{F}_q$
$= (k_{DO} + k_\phi) \cdot c_{ij} + f_{k_{PRF}}(i||j)$
$= (k_{DO} + k_\phi)(\sum_{k=1}^{m} \alpha_{ijk} w_k + \alpha_\pi w_\pi) + f_{k_{PRF}}(i||j)$
$= k_{DO} \sum_{k=1}^{m} \alpha_{ijk} w_k + (k_{DO} + k_\phi)\alpha_\pi w_\pi + f_{k_{PRF}}(i||j)$ //because $k_\phi w_k = 0$

Note that because only $\mathcal{DO}$ knowns k_ϕ, only $\mathcal{DO}$ can generate v_π such that $k_\phi v_\pi = 0$. In other words, if v_π is not generated by $\mathcal{DO}$ but another one (i.e., the user), $k_\phi v_\pi \neq 0$.

In the check phase, $\mathcal{DO}$ verifies the servers using k_{DO} and k_{PRF} with Eq. 14 as follows:

$t'_i = k_{DO} c_i + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u)$
$= \sum_{u=1}^{s} k_{DO} \beta_u c_{ib_u} + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u)$
$= \sum_{u=1}^{s} \beta_u (k_{DO} c_{ib_u} + f_{k_{PRF}}(i||b_u))$
$= \sum_{u=1}^{s} \beta_u (k_{DO} (\sum_{k=1}^{m} \alpha_{ib_u k} w_k + \alpha_\pi w_\pi) + f_{k_{PRF}}(i||b_u))$
$= \sum_{u=1}^{s} \beta_u (k_{DO} \sum_{k=1}^{m} \alpha_{ib_u k} w_k + k_{DO} \alpha_\pi w_\pi + f_{k_{PRF}}(i||b_u))$
$t_i = \sum_{u=1}^{s} \beta_u t_{ib_u}$
$= \sum_{u=1}^{s} \beta_u (k_{DO} \sum_{k=1}^{m} \alpha_{ijk} w_k + (k_{DO} + k_\phi)\alpha_\pi w_\pi + f_{k_{PRF}}(i||j))$
// replace $t_{ib_u} = t_{ij}$ where $j = b_u$

It is clear that $t_i \neq t'_i$. Therefore, $\mathcal{U}$ cannot forge the tag for v_π. □

7.2 Attacks From Servers

Response Replay Attack. Suppose the malicious server performing this attack is $\mathcal{S}_i$.

- Epoch 1: $\mathcal{DO}$ challenges S_i by $Q = \{(b_1, \beta_1), \cdots, (b_s, \beta_s)\}$. S_i responds a valid pair of $\{c_i, t_i\}$. $\mathcal{DO}$ verifies $t_i = k_{DO} c_i + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u)$.
- Epoch 2: $\mathcal{DO}$ challenges S_i by $Q' = \{(b'_1, \beta'_1), \cdots, (b'_s, \beta'_s)\}$. S_i reuses $\{c_i, t_i\}$, but cannot pass the verification because: $t_i \neq k_{DO} c_i + \sum_{u=1}^{s} \beta'_u \cdot f_{k_{PRF}}(i||b'_u)$.

Thus, $\mathcal{S}_i$ cannot pass the verification with this attack. Similarly, in the retrieve phase, $\mathcal{U}$ also performs as the check phase using his keys k_U and k_{PRF}, $\mathcal{S}_i$ also cannot pass the verification with this attack.

Pollution Attack. Suppose that the malicious server S_i provides a polluted pair of $\{c_i, t_i\}$ to $\mathcal{DO}$ (in repair phase) or to $\mathcal{U}$ (in the retrieve phase). The key point here is that $\mathcal{DO}$ and $\mathcal{U}$ always check every provided response. S_i cannot pass the verification $t_i = k_{DO} c_i + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u)$ (Eq. 14) and $t_i = k_U c_i + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u)$ (Eq. 18) if c_i and t_i are not the independent linear combinations of the coded blocks and tags at the points: $\{b_1, \cdots, b_s\}$ using the coefficient $\{\beta_1, \cdots, \beta_s\}$.

8 Efficiency Analysis and Performance Evaluation

Efficiency Analysis. The efficiency comparison between the RDC-NC [20], NC-Audit [22], MD-POR [23] and POR-2P schemes is given in Table 1. The NC-Audit and MD-POR schemes support the public authentication which means that an entity called Third Party Auditor (TPA) is delegated the task of checking the servers by the data owner. The MD-POR has multiple data owners. For the fair comparison, we suppose that the check phase in the NC-Audit and MD-POR schemes is performed by the data owner, and we suppose that the MD-POR only has a single data owner.

Table 1: Comparison

		RDC-NC [20]	NC-Audit [22]	MD-POR [23]	POR-2P (proposal)
Encode phase	Comp. ($\mathcal{DO}$)	$O(mnd)$	$O(mnd)$	$O(m)$	$O(mnd)$
	Comp. (servers)	$O(1)$	$O(1)$	$O(mnd)$	$O(1)$
	Comm.	$O(nd(\frac{\lvert F\rvert}{m}+m))$	$O(nd(\frac{\lvert F\rvert}{m}+m)+mnd)$	$O(mn(\frac{\lvert F\rvert}{m}+m))$	$O(nd(\frac{\lvert F\rvert}{m}+m)$
Check phase	Comp. ($\mathcal{DO}$)	$O(nds)$	$O(ns)$	$O(n)$	$O(n)$
	Comp. (servers)	$O(ndm)$	$O(ns)$	$O(nd)$	$O(ns)$
	Comm.	$O(n(\frac{\lvert F\rvert}{m}+m))$	$O(n(\frac{\lvert F\rvert}{m}+m))$	$O(n(\frac{\lvert F\rvert}{m}+m))$	$O(n(\frac{\lvert F\rvert}{m}+m))$
Repair phase	Comp. ($\mathcal{DO}$)	$O(ld)$	$O(ld)$	$O(ld)$	$O(sd)$
	Comp. (servers)	$O(ld)$	$O(ld)$	$O(ld)$	$O(sd)$
	Comm.	$O((l+d)(\frac{\lvert F\rvert}{m}+m))$	$O((l+d)(\frac{\lvert F\rvert}{m}+m)+ld)$	$O((l+d)(\frac{\lvert F\rvert}{m}+m))$	$O((s+d)(\frac{\lvert F\rvert}{m}+m))$
Retrieve phase	Comp. ($\mathcal{U}$)	N/A	N/A	N/A	$O(sm)+O(G)$
	Comp. ($\mathcal{DO}$)	$O(sm)+O(G)$	$O(sm)+O(G)$	$O(sm)+O(G)$	$O(sm)+O(G)$
	Comp. (servers)	$O(sm)$	$O(sm)$	$O(sm)$	$O(sm)$
	Comm.	$O(\lvert F\rvert+m^2)$	$O(\lvert F\rvert+m^2)$	$O(\lvert F\rvert+m^2)$	$O(\lvert F\rvert+m^2)$

l denotes the number of healthy servers used in the RDC-NC, NC-Audit and MD-POR.
$O(G)$ denotes the complexity of the Gaussian elimination to solve m file blocks.
Comp. means computation.
Comm. means communication.
N/A means not applicable due to the lack of support.

Performance Evaluation. We evaluate the computation performance of the POR-2P scheme to show that it is applicable for a real system. A program written by Python 2.7.3 is executed using a computer with Intel Core i5 processor, 2.4 GHz, 4 GB of RAM, Windows 7 64-bit OS. The prime q is set to be 256 bits. The number of servers is set to be 10 ($n = 10$). The number of coded blocks stored in each server is set to be 20 ($d = 20$). The number of spot checks for each server is set to be a half of d ($s = d/2 = 10$). The size of each file block is set to be 2^{23} bits (1MB). The experiment results are observed with four sets of computation performance: Fig. 3 (encode), Fig. 4 (check), Fig. 5 (repair), and Fig. 6 (retrieve), by varying the file size.

Fig. 3 reveals that the computation time in the encode and init (splitting file) functions linearly increase while the keygen function is constant. If the file size is 1 GB, the computation time in the encode and init functions is roughly 8308.561s and 1863.914s, respectively. Fig. 4 reveals that the computation time of the challenge, respond, and verify functions is constant. Fig. 5 reveals that the computation time of the repair function is also constant. Fig. 6 reveals that the computation time in the retrieve function is almost linear in user-side and is constant in server-side. If the file size is 1GB, the computation time of retrieve function in user-side is roughly 2691.186s.

Although the graphs of the encode, the init and the retrieve functions on data owner-side linearly increase with the file size, the encode and init phases are executed only one time in the beginning; and the retrieve phase is also executed very few. Meanwhile, the check and repair phases are executed very often during the system lifetime with a constant computation as showed in the graphs. The above results show that the POR-2P is very efficient and applicable in a real system.

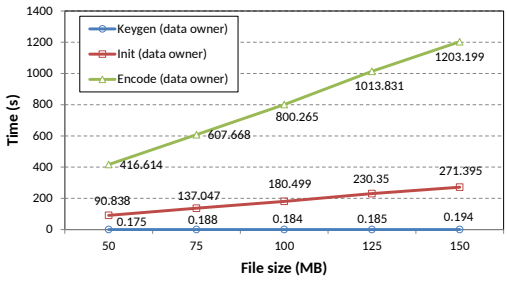

Fig. 3: Computation of encode phase

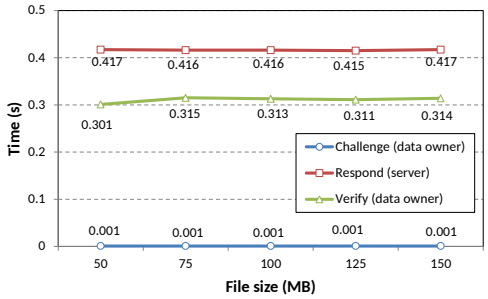

Fig. 4: Computation of check phase

Fig. 5: Computation of repair phase

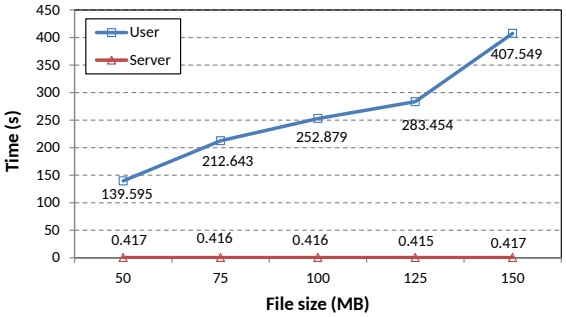

Fig. 6: Computation of retrieve phase

9 Conclusion and Future Work

This paper proposes the POR-2P to support the data provision-payment system in which the user can check and can retrieve the data using a symmetric key setting. The POR-2R can also deal with access control during retrieval process. The security analysis shows that the POR-2P can prevent the attacks from the user and the servers. Furthermore, the efficiency analysis and performance evaluation show that POR-2P is very applicable to a real system. Further work is investigated the implementation of the previous schemes.

References

1. A. Juels and B. Kaliski: PORs: Proofs of retrievability for large files, *Proc. 14th ACM Conf. on Computer and Communications Security - CCS'07*, pp. 584-597, 2007.
2. H. Shacham and B. Waters: Compact Proofs of Retrievability, *Proc. 14th Conf. on the Theory and Application of Cryptology and Information Security - ASIACRYPT'08*, pp. 90-107, 2008.
3. K. Bowers, A. Juels and A. Oprea: Proofs of retrievability: theory and implementation, *Proc. Workshop on Cloud computing security - CCSW'09*, pp. 43-54, 2009.
4. W. J. Bolosky, J. R. Douceur, D. Ely and M. Theimer: Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs, *Proc. of ACM conf. on Measurement and modeling of computation systems - SIGMETRICS'00*, pp. 34-43, 2000.
5. R. Curtmola, O. Khan, R. Burns and G. Ateniese: MR-PDP: Multiple-Replica Provable Data Possession, *Proc. 28th Conf. on Distributed Computing Systems*, pp. 411-420, 2008.
6. M. K. Aguilera, R. Janakiraman and L. Xu: Efficient fault-tolerant distributed storage using erasure codes, *Technical Report, Washington University in St. Louis*, 2004.
7. K. Bowers, A. Juels and A. Oprea: HAIL: A High-Availability and Integrity Layer for Cloud Storage, *Proc. 16th ACM Conf. on Computer and Communications Security - CCS'09*, pp. 187-198, 2009.
8. E. Shi, E. Stefanov, and C. Papamanthou: Practical dynamic proofs of retrievability, *Proc. of ACM SIGSAC conf. on Computer and Communications Security - CCS'13*, pp. 325-336 (2013).

9. D. Cash, A. Kupcu, and D. Wichs: Dynamic Proofs of Retrievability via Oblivious RAM, *Proc. of 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT'13*, vol. 7881, pp. 279-295, 2013.
10. R. Ahlswede, N. Cai, S. Li and R. Yeung: Network information flow, *IEEE Trans. on Information Theory*, 46(4):1204-1216, 2000.
11. S.Y.R. Li, R.W. Yeung, and N. Cai: Linear Network Coding, *IEEE Trans. on Information Theory*, 49(2):371-381, 2003.
12. A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright and K. Ramchandran: Network coding for distributed storage systems, *IEEE Trans. on Information Theory*, 56(9):4539-4551, 2010.
13. S. Acedanski, S. Deb, M. Medard, and R. Koetter, "How good is random linear coding based distributed networked storage?", *Workshop on Network Coding, Theory and Applications - NETCOD'05* (2005).
14. S. Agrawal and D. Boneh: Homomorphic MACs: MAC-Based Integrity for Network Coding, *Proc. 7th Conf. on Applied Cryptography and Network Security - ACNS'09*, pp. 292-305, 2009.
15. C. Cheng and T. Jiang: An Efficient Homomorphic MAC with Small Key Size for Authentication in Network Coding, *IEEE Trans. on Computers*, 62(10):2096-2100, 2012.
16. C. Cheng, T. Jiang and Q. Zhang: TESLA-Based Homomorphic MAC for Authentication in P2P System for Live Streaming with Network Coding, *IEEE Journal on Selected Areas in Communications*, 31(9):291-298, 2013.
17. R. Johnson, D. Molnar, D. Song and D Wagner: Homomorphic Signature Schemes, *Proc. of Cryptographer's Track at the RSA Conf. on Topics in Cryptology - CT-RSA'02*, pp. 244-262, 2002.
18. D.M. Freeman: Improved security for linearly homomorphic signatures: a generic framework, *Proc. of 15th conf. on Practice and Theory in Public Key Cryptography - PKC'12*, vol.7293, pp. 697-714, 2012.
19. J. Li, S. Yang, X. Wang, X. Xue and B. Li: Tree-structured Data Regeneration in Distributed Storage Systems with Network Coding, *Proc. 29th conference on Information communications - INFOCOM'10*, pp. 2892-2900, 2010.
20. B. Chen, R. Curtmola, G. Ateniese and R. Burns: Remote Data Checking for Network Coding-based Distributed Storage Systems, *Proc. ACM Cloud Computing Security Workshop - CCSW'10*, pp. 31-42, 2010.
21. H.C.H. Chen, Y. Hu, P.P.C. Lee and Y. Tang: NCCloud: A Network-Coding-Based Storage System in a Cloud-of-Clouds, *IEEE Trans. on Computers*, 63(1):31-44, 2014.
22. A. Le and A. Markopoulou: NC-Audit: Auditing for network coding storage, *International Symposium on Network Coding - NetCod'12*, pp. 155-160, 2012.
23. K. Omote, T.P. Thao: MD-POR: Multi-source and Direct Repair for Network Coding-based Proof of Retrievability, *International Journal of Distributed Sensor Networks (IJDSN)*, vol. 2015, article ID: 586720, Jan 2015.
24. W. Yan, M. Yang, L. Li, and H. Fang: Short signature scheme for multi-source network coding, *Computer Communications*, 35(3):344-351, 2012.
25. R. Koetter and M. Medard: An algebraic approach to network coding, *IEEE/ACM Trans. on Networking*, 11(5):782-795, 2003.
26. A. Le and A. Markopoulou: On detecting pollution attacks in inter-session network coding, *Proc. 31st IEEE conference on Computer Communications - INFOCOM'12*, pp. 343-351, 2012.