IEICE TRANS. INF. & SYST., VOL.E99–D, NO.4 APRIL 2016

PAPER *Special Section on Information and Communication System Security*

D2-POR: Direct Repair and Dynamic Operations in Network Coding-Based Proof of Retrievability*

Kazumasa OMOTE†, ***Member and*** **Phuong-Thao TRAN**†a), ***Nonmember***

SUMMARY Proof of Retrievability (POR) is a protocol by which a client can distribute his/her data to cloud servers and can check if the data stored in the servers is available and intact. After that, network coding-based POR has been applied to improve network throughput. Although many network coding-based PORs have been proposed, most of them have not achieved the following practical features: direct repair and dynamic operations. In this paper, we propose the D2-POR scheme (Direct repair and Dynamic operations in network coding-based POR) to address these shortcomings. When a server is corrupted, the D2-POR can support the direct repair in which the data stored in the corrupted server can be repaired using the data directly provided by healthy servers. The client is thus free from the burden of data repair. Furthermore, the D2-POR allows the client to efficiently perform dynamic operations, i.e., modification, insertion and deletion.

key words: *Proof of Retrievability, network coding, direct repair, dynamic operations*

1. Introduction

Since amount of data is increasing exponentially, data storage and data management become burdensome tasks of the client. Therefore, storage providers called clouds have been proposed to allow the client to store, manage and share the data portably and easily from anywhere via the Internet. However, because cloud providers could not be trustworthy, this system introduces three security challenges: data availability, data integrity and data confidentiality. Ensuring data availability and data integrity is the primary requirement before ensuring data confidentiality because data availability and data integrity are the prerequisites of the existence of a system. This paper thus focuses on data availability and data integrity. To support the client to check whether the data stored in the servers is available and intact, researchers proposed Proof of Retrievability (POR) [1]–[3], which is a challenge-response protocol between a client and a server. Based on the POR, the following three approaches are commonly used: replication, erasure coding, and network coding. In the replication [4]–[6], the client stores a file replica in each server. The client can perform periodic server checks. If a server is corrupted, the client will use a healthy replica to repair the corruption. The drawback of this approach is the high storage cost for the redundant replicas. To address this drawback, erasure coding has been applied in [7]–[10]. Instead of storing file replicas as replication, the client stores file blocks in each server. Hence, the storage cost can be reduced. However, the drawback of this approach is that to repair a corrupted server, the client must reconstruct the original file before generating new coded blocks. The computation cost is thus increased during data repair. To address this drawback, network coding has been applied in [11]–[13] in which the client does not need to reconstruct the original file before repairing the corruption. Instead, the client can retrieve coded blocks from healthy servers to generate new coded blocks. Therefore, this paper focuses on network coding. In addition, the data stored in the servers cannot be checked without additional information, i.e., Message Authentication Code (MAC tag) (used in a symmetric key setting) or signature (used in an asymmetric key setting). Because it is well-known that a symmetric key setting is more efficient than an asymmetric key setting, we thus focus on MAC tags. Concretely, to be suitable for network coding, we use homomorphic MACs in our scheme [14]–[16].

Network Coding-based POR. Dimakis et al. [17] were the first to apply the network coding to distributed storage systems and achieve a reduction in the communication overhead of the repair component. Li et al. [18] introduced a tree-structure data regeneration with linear network coding to achieve an efficient regeneration traffic and bandwidth capacity by using an undirected-weighted maximum spanning tree. Chen et al. [19] then proposed the Remote Data Checking for Network Coding-based distributed storage system (RDC-NC) which provides an elegant data repair by recoding encoded blocks in healthy servers during repair. H. Chen et al. [20] proposed the NC-Cloud scheme to improve cost-effective repair using the functional minimum-storage regenerating (FMSR) code and lighten the encoding requirement of storage nodes during repair. However, most of these schemes have the following shortcomings. Firstly, the schemes can only support the indirect repair. That is, to repair a corrupted server, the client must require the healthy servers to provide aggregated coded blocks and aggregated tags, and send them back to the client. The client then checks the provided coded blocks using the provided tags, and computes new coded blocks and new tags to replace the corruption. The client sends the new coded blocks and tags

Manuscript received May 29, 2015.
Manuscript revised August 21, 2015.
Manuscript publicized January 13, 2016.

†The authors are with Japan Advanced Institute of Science and Technology, Nomi-shi, 923–1292 Japan.

*The preliminary version of this paper was presented at COCOON'15 [28]. This study is partly supported by Grant-in-Aid for Young Scientists (B) (25730083).

a) E-mail: tpthao@jaist.ac.jp

DOI: 10.1587/transinf.2015ICP0014

to the new server. Such a repair mechanism is a troublesome task for the client. Because the data repair is performed very often during the system lifetime, the client thus incurs high computation and communication costs. Secondly, the schemes do not consider dynamic operations. That is, the client can only perform data check and data retrieval, but cannot perform modification, insertion and deletion. A few PORs were proposed to deal with the dynamic operations, e.g, [21]–[24]; however, all these schemes are based on erasure coding, not network coding.

There are two notable schemes which are mostly related to our proposed scheme. The first one is the MD-POR [25], which can support the direct repair, but cannot support the dynamic operations. The second one is the NC-Audit [26], which also considered the direct repair and dynamic operations. However, when the direct repair is supported, this scheme cannot prevent pollution attack which is a common attack of network coding. This is because the new server cannot check the provided coded blocks. Furthermore, the dynamic operations have not been completed and have not been discussed with clear details. For the insertion, the authors said that the insertion does not work in their scheme. For the modification, the authors discuss how to update tags without showing how to update coded blocks. For the deletion, there is no concrete explanation.

Contribution. In this paper, we propose the D2-POR scheme with the following contributions:

- Direct repair: when a server is corrupted, the healthy servers will provide their coded blocks and tags directly to the new server without sending them back to the client. Then, the new server can check them to prevent pollution attack, and can compute new coded blocks and tags for itself. The client is thus free from the repair process.
- Dynamic operations: the client not only can check and retrieve the data, but also can modify, insert and delete the data.
- Symmetric key setting: our scheme does not use any public key for the efficiency. The direct repair feature introduces a challenge that how to allow the new server which is untrusted to check and compute new coded blocks and tags without using a public key. Our scheme can address this problem by using an orthogonal key technique called InterMac [27].

Roadmap. The backgrounds of the POR, network coding and InterMac are described in Sect. 2. The adversarial model is presented in Sect. 3. The D2-POR scheme is proposed in Sect. 4. The security and efficiency analyses are shown in Sects. 5 and 6. A numeric example is given in Sect. 7. The conclusion is drawn in Sect. 8.

2. Background

2.1 The POR Framework

The POR [1]–[3] is a challenge-response protocol between a verifier V (client) and a prover P (server), and consists of the following algorithms:

- $\mathsf{Keygen}(\lambda) \rightarrow \kappa$: run by V. This algorithm inputs a security parameter s and outputs a secret key κ (For an asymmetric key system, κ is a public/private key pair.)
- $\mathsf{Encode}(F, \kappa) \rightarrow F'$: run by V. This algorithm inputs an original file F and κ, and outputs an encoded file F^*, and then sends F^* to P.
- $\mathsf{Check}() \rightarrow \{\mathsf{accept/deny}\}$: run by both V and P. Firstly, V generates a challenge c and sends c to P. P then computes a response r and sends r back to V. V finally verifies P based on c and r.
- $\mathsf{Repair}()$: run by V. When a corruption is detected, V executes this algorithm to repair the corruption. The technique of repair depends on each specific scheme, i.e, replication, erasure coding or network coding.

2.2 Network Coding in Distributed Storage System

Network coding [11]–[13] has been proposed for cost-efficiency in data transmission and data repair. The model system consists of a client and multiple servers. The client owns a file F and wants to redundantly store coded blocks in the servers in a way that the client can reconstruct F and can repair coded blocks in a corrupted server. The client firstly divides F into m blocks: $F = v_1 \| \cdots \| v_m$. Each $v_k \in \mathbb{F}_q^z$ where $k \in \{1, \cdots, m\}$. $\mathbb{F}_q^z$ denotes a z-dimensional finite field of a prime order q. The client then augments v_k with a vector of length m which contains a '1' bit in the k-th position and $(m-1)$ '0' bits elsewhere. The resulting block is called *augmented block* (denoted by 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 client randomly chooses m coefficients $\alpha_1, \cdots, \alpha_m$ in $\mathbb{F}_q$ to compute coded blocks using the linear combination $c = \sum_{k=1}^{m} \alpha_k \cdot w_k \in \mathbb{F}_q^{z+m}$. The clients stores the 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 all file blocks. Note that the matrix consisting of the coefficients used to construct any m coded blocks should have full rank. Koetter et al. [13] 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 client repairs it by retrieving the coded blocks

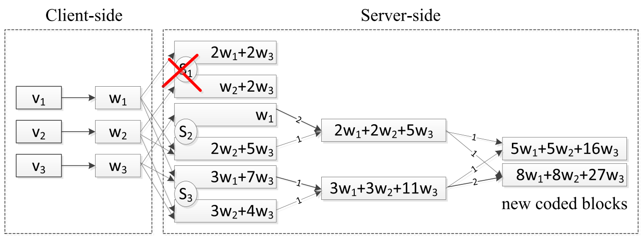

Fig. 1 The client stores the coded blocks in the server $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$. Suppose that $\mathcal{S}_1$ is corrupted, the client repair it by the linear combinations of the coded blocks from $\mathcal{S}_2$ and $\mathcal{S}_3$.

from healthy servers and linearly combining them to regenerate new coded blocks. An example of the data repair is given in Fig. 1.

2.3 InterMac

In our scheme, the direct repair yields a challenge that how to allow the new server which is untrusted to check the provided coded blocks without learning the secret key of the client. The InterMac [27] is a suitable technique to generate such a key for the new server. Basically, the InterMac is proposed to generate a vector which is orthogonal to a given set of vectors. Formally, given a set of vectors $\{w_1, \cdots, w_m\}$, the algorithm outputs a vector k_p such that $w_k \cdot k_p = 0$, $\forall k \in \{1, \cdots, m\}$.

1. The InterMac algorithm is described as follows:
   $\mathsf{InterMac}(w_1, \cdots, w_m) \to k_p$:
   - Find the span π of $w_1, \cdots, w_m \in \mathbb{F}_q^{z+m}$.
   - Construct matrix M in which $\{w_1, \cdots, w_m\}$ are the rows of M.
   - Find the null-space of M, denoted by $\pi_M^{\perp}$, which is the set of all vectors $u \in \mathbb{F}_q^{z+m}$ such that $M \cdot u^T = 0$.
   - Find the basis vectors of $\pi_M^{\perp}$, denoted by $B_1, \cdots, B_z \in \mathbb{F}_q^{z+m}$ // Theorem 1 will explain why the number of basis vectors is z.
   - Compute $k_p \leftarrow \mathsf{Kg}(B_1, \cdots, B_z)$.
2. The sub-algorithm Kg used in the InterMac algorithm is given as follows:
   $\mathsf{Kg}(B_1, \cdots, B_z) \to k_p$:
   - Generate $r_i \overset{rand}{\leftarrow} \mathbb{F}_q$.
   - Compute $k_p \leftarrow \sum_{i=1}^{z} r_i \cdot B_i \in \mathbb{F}_q^{z+m}$.

Theorem 1: Given $\{w_1, \cdots, w_m\}$ (each $w_k \in \mathbb{F}_q^{z+m}$), the number of basis vectors of $\pi_M^{\perp}$ is z.

Proof: $\mathsf{rank}(M) = m$. Let π_M be the space spanned by the rows of M. For any $m \times (z+m)$ matrix, the rank-nullity theorem gives: $\mathsf{rank}(M) + \mathsf{nullity}(M) = z + m$ where $\mathsf{nullity}(M)$ is the dimension of $\pi_M^{\perp}$. It yields: $\dim(\pi_M^{\perp}) = (z + m) - m = z$. Therefore, the number of basis vectors of $\pi_M^{\perp}$ is z. In the InterMac, we denote the basis vectors by $B_1, \cdots, B_z$. □

3. Adversarial Model

In our scheme, the client is trusted and the servers are untrusted. Assume that the servers do not collude with each other. The servers can perform two types of attacks:

1. In the check phase: the servers disrupts the system or modifies the data. The attacks can be commonly prevented by the tags, we thus do not focus on them.
2. In the repair phase: the servers can perform: (i) pollution attack which is a common attack of network coding, and (ii) curious attack which is a special attack of the direct repair. We focus on them in the security analysis.
   - *Pollution Attack.* A malicious server firstly uses a valid coded block to pass the check phase, but then injects an invalid coded block in the repair phase to prevent data repair. An example is given as follows:
     – Encode: the client encodes augmented blocks (w_1, w_2, w_3) to six coded blocks: c_{11}, c_{12} (stored in the server $\mathcal{S}_1$), c_{21}, c_{22} (stored in the server $\mathcal{S}_2$), and c_{31}, c_{32} (stored in the server $\mathcal{S}_3$). Suppose that $\mathcal{S}_1$ will perform a pollution attack.
     – Check: $\mathcal{S}_3$ is corrupted.
     – Repair: $\mathcal{S}_3$ should be repaired by two coded blocks: c'_{31} (which is a linear combination of c_{11} and c_{12}) and c'_{32} (which is a linear combination of c_{21} and c_{22}). However, $\mathcal{S}_1$ is not detected because this time is the repair phase, not the check phase. The client still thinks $\mathcal{S}_1$ is healthy and requests coded blocks from $\mathcal{S}_1$ and $\mathcal{S}_2$. $\mathcal{S}_1$ will provide an invalid coded block c''_{31} to the client instead of c'_{31}.
   - *Curious Attack.* This attack is performed by the new server in the repair phase. Every repair time, the new server is given a key k_r constructed from the secret key k_C of the client and a variant k_p. Having k_r, the new server tries to obtain k_C in order to pass the check phases in the later time step (called epoch).

4. Proposed D2-POR Scheme

4.1 Notations

The notations used throughout the D2-POR scheme are given in Table 1.

4.2 Construction

4.2.1 Setup

(1) *Create augmented blocks*: C divides F into m blocks $F = v_1 \| \cdots \| v_m$. Each $v_k \in \mathbb{F}_q^z$ where $k \in \{1, \cdots, m\}$. C creates m augmented blocks as Eq. (1).

Table 1 Notations

C	client
F	original file
m	number of file blocks
n	number of servers
d	number of coded blocks in each server
k	file block index ($k \in \{1, \cdots, m\}$)
i	server index ($i \in \{1, \cdots, n\}$)
j	coded block index in each server ($j \in \{1, \cdots, d\}$)
v_k	file block ($k \in \{1, \cdots, m\}$)
w_k	augmented block of v_k
t_{w_k}	tag of w_k
$\mathcal{S}_i$	server
c_{ij}	j-th coded block stored in $\mathcal{S}_i$
t_{ij}	tag of c_{ij}
l	number of healthy servers used for data repair
$\mathcal{S}_r$	corrupted server
$\mathcal{S}'_r$	new server which is used to replace $\mathcal{S}_r$
$\mathbb{F}_q^z$	z-dimensional finite field $\mathbb{F}$ of a prime order q

(2) *Keygen*: C generates two types of keys:

- *Key of the client* (k_C): $k_C \overset{rand}{\leftarrow} \mathbb{F}_q^{z+m}$.
- *One-time key for a new server every repair time* (k_r): C firstly computes a value $k_p \in \mathbb{F}_q^{z+m}$ such that $w_k \cdot k_p = 0$ for $\forall k \in \{1, \cdots, m\}$ by using $k_p \leftarrow \mathsf{InterMac}(w_1, \cdots, w_m)$. C then computes:

$$k_r = k_C + k_p \in \mathbb{F}_q^{z+m} \tag{2}$$

k_C is static, only k_p is recomputed every repair time. k_r is sent to the new server only when a data repair is executed.

4.2.2 Encode

(1) C computes a tag for each augmented block:
For $\forall k \in \{1, \cdots, m\}$:

$$t_{w_k} = w_k \cdot k_C \in \mathbb{F}_q \tag{3}$$

(2) C computes nd coded blocks and nd corresponding tags as follows:
For $\forall i \in \{1, \cdots, n\}, \forall j \in \{1, \cdots, d\}$:

- C generates m coefficients: $\alpha_{ijk} \overset{rand}{\leftarrow} \mathbb{F}_q$ where $k \in \{1, \cdots, m\}$.
- C computes code blocks:

$$c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} \cdot w_k \in \mathbb{F}_q^{z+m} \tag{4}$$

- C compute tags:

$$t_{ij} = \sum_{k=1}^{m} \alpha_{ijk} \cdot t_{w_k} \in \mathbb{F}_q \tag{5}$$

(3) C sends $\{c_{ij}, t_{ij}\}$ where $j \in \{1, \cdots, d\}$ to server $\mathcal{S}_i$.

4.2.3 Check

(1) C requires $\mathcal{S}_i$ to provide its proof.
(2) $\mathcal{S}_i$ where $i \in \{1, \cdots, n\}$ combines coded blocks and tags as follows:

- $\mathcal{S}_i$ generates d coefficients: $\beta_{ij} \overset{rand}{\leftarrow} \mathbb{F}_q$ where $j \in \{1, \cdots, d\}$.
- $\mathcal{S}_i$ combines coded blocks:

$$c_{\mathcal{S}_i} = \sum_{j=1}^{d} \beta_{ij} \cdot c_{ij} \in \mathbb{F}_q^{z+m} \tag{6}$$

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

$$t_{\mathcal{S}_i} = \sum_{j=1}^{d} \beta_{ij} \cdot t_{ij} \in \mathbb{F}_q \tag{7}$$

- $\mathcal{S}_i$ sends $\{c_{\mathcal{S}_i}, t_{\mathcal{S}_i}\}$ to C.

(3) C verifies $\mathcal{S}_i$ as follows:
For $\forall i \in \{1, \cdots, n\}$:

- C computes:

$$t'_{\mathcal{S}_i} = c_{\mathcal{S}_i} \cdot k_C \in \mathbb{F}_q \tag{8}$$

- C checks iff:

$$t_{\mathcal{S}_i} = t'_{\mathcal{S}_i} \tag{9}$$

If it holds, $\mathcal{S}_i$ is healthy. Otherwise, $\mathcal{S}_i$ is corrupted.

4.2.4 Repair

Suppose $\mathcal{S}_r$ is corrupted. A set of l healthy servers $\{\mathcal{S}_{i_1}, \cdots, \mathcal{S}_{i_l}\}$ are required to provide data to a new server $\mathcal{S}'_r$, which is used to replace $\mathcal{S}_r$.
(1) $\mathcal{S}_i$ where $i \in \{i_1, \cdots, i_l\}$ provides its data to $\mathcal{S}'_r$ as follows:

- $\mathcal{S}_i$ generates d coefficients: $\beta_{ij} \overset{rand}{\leftarrow} \mathbb{F}_q$ where $j \in \{1, \cdots, d\}$.
- $\mathcal{S}_i$ combines coded blocks:

$$c_{\mathcal{S}_i} = \sum_{j=1}^{d} \beta_{ij} \cdot c_{ij} \in \mathbb{F}_q^{z+m} \tag{10}$$

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

$$t_{\mathcal{S}_i} = \sum_{j=1}^{d} \beta_{ij} \cdot t_{ij} \in \mathbb{F}_q \tag{11}$$

- $\mathcal{S}_i$ sends $\{c_{\mathcal{S}_i}, t_{\mathcal{S}_i}\}$ to $\mathcal{S}'_r$.

(2) $\mathcal{S}'_r$ checks $\mathcal{S}_i$ where $i \in \{i_1, \cdots, i_l\}$ as follows:

- $\mathcal{S}'_r$ computes:

$$t'_{\mathcal{S}_i} = c_{\mathcal{S}_i} \cdot k_r \in \mathbb{F}_q \tag{12}$$

- $\mathcal{S}'_r$ checks iff:

$$t'_{\mathcal{S}_i} = t_{\mathcal{S}_i} \tag{13}$$

(3) $\mathcal{S}'_r$ computes d new coded blocks and d tags:
For $\forall j \in \{1, \cdots, d\}$:

- $\mathcal{S}'_r$ generates l coefficients: $\gamma_{ri} \overset{rand}{\leftarrow} \mathbb{F}_q$ where $i \in \{i_1, \cdots, i_l\}$.
- $\mathcal{S}'_r$ computes new coded block:

$$c_{rj} = \sum_{i=i_1}^{i_l} \gamma_{ri} \cdot c_{\mathcal{S}_i} \in \mathbb{F}_q^{z+m} \tag{14}$$

- $\mathcal{S}'_r$ computes new tag:

$$t_{rj} = \sum\nolimits_{i=i_1}^{i_l} \gamma_{ri} \cdot t_{S_i} \in \mathbb{F}_q \tag{15}$$

4.3 Correctness

(1) The correctness of Eq. (9) is proved as follows:

$$\begin{aligned}
t_{S_i} &= \textstyle\sum_{j=1}^{d} \beta_{ij} t_{ij} \text{//due to Eq. (7)}\\
&= \textstyle\sum_{j=1}^{d}\sum_{k=1}^{m} \beta_{ij}\alpha_{ijk} t_{w_k} \text{// due to Eq. (5)}\\
&= \textstyle\sum_{j=1}^{d}\sum_{k=1}^{m} \beta_{ij}\alpha_{ijk} w_k k_C \text{//due to Eq. (3)}\\
t'_{S_i} &= c_{S_i} k_C \text{ //due to Eq. (8)}\\
&= \textstyle\sum_{j=1}^{d} \beta_{ij} c_{ij} k_C \text{ //due to Eq. (6)}\\
&= \textstyle\sum_{j=1}^{d}\sum_{k=1}^{m} \beta_{ij}\alpha_{ijk} w_k k_C \text{ //due to Eq. (4)}\\
&= t_{S_i}.
\end{aligned}$$

(2) The correctness of Eq. (13) is proved as follows:

$$\begin{aligned}
t_{S_i} &= \textstyle\sum_{j=1}^{d} \beta_{ij} t_{ij} \text{//due to Eq. (11)}\\
&= \textstyle\sum_{j=1}^{d}\sum_{k=1}^{m} \beta_{ij}\alpha_{ijk} t_{w_k} \text{ //due to Eq. (5)}\\
&= \textstyle\sum_{j=1}^{d}\sum_{k=1}^{m} \beta_{ij}\alpha_{ijk} w_k k_C \text{ //due to Eq. (3)}\\
t'_{S_i} &= c_{S_i} k_r \text{ //due to Eq. (12)}\\
&= \textstyle\sum_{j=1}^{d} \beta_{ij} c_{ij} k_r \text{ //due to Eq. (10)}\\
&= \textstyle\sum_{j=1}^{d}\sum_{k=1}^{m} \beta_{ij}\alpha_{ijk} w_k k_r \text{ //due to Eq. (4)}\\
&= \textstyle\sum_{j=1}^{d}\sum_{k=1}^{m} \beta_{ij}\alpha_{ijk} w_k (k_C + k_p) \text{ //due to Eq. (2)}\\
&= \textstyle\sum_{j=1}^{d}\sum_{k=1}^{m} \beta_{ij}\alpha_{ijk} w_k k_C \text{ // due to } k_p w_k = 0\\
&= t_{S_i}
\end{aligned}$$

4.4 Dynamic Operations

When C performs a dynamic operation on a file block, herein introduces a challenge: how the servers deal with the coded blocks which are related to the modified/ inserted/deleted file block. The trivial solution, which is to encode the data again, incurs very high costs. In our solution, the old coded blocks and tags stored in the servers can be re-used, and only a small additional computation is needed for the dynamic operations.

Firstly, we give the following theorem, which will form the basis of the dynamic operations.

Theorem 2: The number of basis vectors of the matrix consisting of m augmented blocks (each augmented block belongs to $\mathbb{F}_q^{z+m}$) is z.

Proof: Let M be the matrix in which m augmented blocks are rows in M:

$$M = \begin{pmatrix} w_1\\ w_2\\ \vdots\\ w_m \end{pmatrix} = \underbrace{\begin{pmatrix} \overbrace{v_1}^{z} & \overbrace{\begin{matrix}1 & 0 & 0 & \cdots & 0\end{matrix}}^{m}\\ v_2 & \begin{matrix}0 & 1 & 0 & \cdots & 0\end{matrix}\\ \vdots & \begin{matrix}\vdots & \vdots & \vdots & \vdots & \vdots\end{matrix}\\ v_m & \begin{matrix}0 & \cdots & \cdots & 0 & 1\end{matrix} \end{pmatrix}}_{m\times(z+m)} \tag{16}$$

Because each augmented block $w_k \in \mathbb{F}_q^{z+m}$ (where $k \in \{1,\cdots,m\}$) consists of $v_k \in \mathbb{F}_q^{z}$ and m elements in $\mathbb{F}_q$, the dimension of M is $m \times (z+m)$. Thus, the number of pivot variables is m. The number free variables is $(z+m)-m = z$. Therefore, the number of basis vectors of M is z. □

4.4.1 Modification

Suppose that C modifies a file block v_X to a new file block v'_X where $X \in \{1,\cdots,m\}$. Let w_X and w'_X be the augmented block of v_X and v'_X, respectively.

(1) *C modifies k_r:*

Let M be the matrix consisting of m augmented blocks. After the modification, only v_X is changed and other elements in M are unchanged. Namely, M is changed to M' as follows:

$$M = \underbrace{\begin{pmatrix} v_1 & 1 & 0 & 0 & \cdots & \cdots & \cdots & 0\\ v_2 & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ \boxed{v_X} & \multicolumn{4}{l}{\underbrace{0 \;\cdots\; 0 \;\; 1}_{X}} & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ v_m & 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}}_{m\times(z+m)}$$

$$M' = \underbrace{\begin{pmatrix} v_1 & 1 & 0 & 0 & \cdots & \cdots & \cdots & 0\\ v_2 & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ \boxed{v'_X} & \multicolumn{4}{l}{\underbrace{0 \;\cdots\; 0 \;\; 1}_{X}} & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ v_m & 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}}_{m\times(z+m)}$$

The modification does not affect k_C because $k_C \overset{rand}{\leftarrow} \mathbb{F}_q^{z+m}$. However, the modification will affect $k_r (= k_C + k_p)$ because k_p is constructed from M. This is why we need to update k_r.

The number of columns in M is $(z+m)$. The number of basis vectors of M is z (Theorem 2). Thus, each of these z basis vectors consists of $(z+m)$ elements in $\mathbb{F}_q$, denoted by $B_\psi = (b_1,\cdots,b_{z+m})^T$ where $\psi \in \{1,\cdots,z\}$. Similarly, each of the z basis vectors of M' also consists of $(z+m)$ elements in $\mathbb{F}_q$, denoted by $B'_\psi = (b'_1,\cdots,b'_{z+m})^T$ where $\psi \in \{1,\cdots,z\}$. We need to find B'_ψ from B_ψ.

Because $v_X \in \mathbb{F}_q^z$, v_X is viewed as a vector of z elements in $\mathbb{F}_q$: $v_X = (v_{X1},\cdots,v_{Xz})$. In M, only v_X is changed and other elements are unchanged. Thus, for each $\psi \in \{1,\cdots,z\}$, C only needs to update the $(z+X)$-th element of B_ψ by computing $((-\sum_{\mu=1}^{z} v'_{X\mu} b_\mu) \mod q)$. Namely:

$$B'_\psi = (b_1,\cdots,b_{z+X-1}, \boxed{(-\sum_{\mu=1}^{z} v'_{X\mu} b_\mu) \mod q}, b_{z+X+1},\cdots,b_{z+m})^T \tag{17}$$

After having B'_ψ for $\psi \in \{1,\cdots,z\}$, C computes $k'_p \leftarrow \mathsf{Kg}(B'_1,\cdots,B'_z)$ (Sect. 2.3). C finally sends $k'_r = k_C + k'_p$ to a new server when a next repair phase is executed.

(2) *C computes tag for w'_k*:

- C computes: $t'_X = w'_X k_C \in \mathbb{F}_q$
- C sends $\{w'_X, t'_X\}$ to $\mathcal{S}_i$.

(3) *$\mathcal{S}_i$ updates coded blocks and tags*:

Because $v_k \in \mathbb{F}_q^z$, v_k can be viewed as a vector of z elements in $\mathbb{F}_q$: $v_k = (v_{k1}, \cdots, v_{kz})$. An augmented block $w_k \in \mathbb{F}_q^{z+m}$ has the form:

$$w_k = (v_{k1}, \cdots, v_{kz}, \overbrace{\underbrace{0, \cdots, 0, 1}_{k}, 0, \cdots, 0}^{m}) \in \mathbb{F}_q^{z+m} \tag{18}$$

Because a coded block $c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} w_k \in \mathbb{F}_q^{z+m}$, c_{ij} can be also viewed as a vector of $(z+m)$ elements in $\mathbb{F}_q$. Let $c_{ij}[x]$ denote the x-th element of c_{ij} where $x \in \{1, \cdots, z+m\}$:

$$c_{ij} = \begin{pmatrix} \sum_{k=1}^{m} \alpha_{ijk} v_{k1} \\ \vdots \\ \sum_{k=1}^{m} \alpha_{ijk} v_{kz} \\ \alpha_{ij1} \\ \vdots \\ \alpha_{ijm} \end{pmatrix}^T = \begin{pmatrix} c_{ij}[1] \\ \vdots \\ c_{ij}[z] \\ c_{ij}[z+1] \\ \vdots \\ c_{ij}[z+m] \end{pmatrix}^T \tag{19}$$

For $\forall j \in \{1, \cdots, d\}$, $\mathcal{S}_i$ computes new coded blocks:

$$c'_{ij} = \begin{pmatrix} c_{ij}[1] + \alpha_{ijX}(v'_{X1} - v_{X1}) \\ \vdots \\ c_{ij}[z] + \alpha_{ijX}(v'_{Xz} - v_{Xz}) \\ c_{ij}[z+1] \\ \vdots \\ c_{ij}[z+m] \end{pmatrix}^T \tag{20}$$

For $\forall j \in \{1, \cdots, d\}$, $\mathcal{S}_i$ computes new tags:

$$t'_{ij} = t_{ij} + \alpha_{ijX}(t'_X - t_X) \tag{21}$$

where c_{ij} and t_{ij} are old coded blocks and tags. Coefficient α_{ijX} can be found at the $(z+X)$-th element of c_{ij}.

The modification only needs $O(z)$ for modifying k_r, $O(1)$ for computing tag for w'_k and $O(z)$ for updating a coded block and a tag.

4.4.2 Insertion

Suppose that C inserts a file block v_I after an existing file block v_X where $X \in \{1, \cdots, m\}$. Let w_I be the augmented block of v_I.

(1) *C modifies k_C*:

Before the insertion, w_k has $(z+m)$ elements in $\mathbb{F}_q$ as Eq. (18). Thus, k_C has $(z+m)$ elements in $\mathbb{F}_q$ (says, $k_C = (k_1, \cdots, k_{z+m})^T$). After the insertion, w_k has $(z+m+1)$ elements in $\mathbb{F}_q$:

$$w_k = (v_{k1}, \cdots, v_{kz}, \overbrace{\underbrace{0, \cdots, 0, 1}_{k}, 0, \cdots, 0}^{m+1}) \in \mathbb{F}_q^{z+m+1}$$

Thus, new k'_C also has $(z+m+1)$ elements in $\mathbb{F}_q$ (says, $k'_C = (k'_1, \cdots, k'_{z+m+1})^T$). Given k_C, we find k'_C:

- The first $(z+X)$ elements of k'_C are the same as the first $(z+X)$ elements of k_C.
- The $(z+X+1)$-th element of k'_C (denoted by k_I) is computed as: $k_I \overset{rand}{\leftarrow} \mathbb{F}_q$.
- The last $(m-X)$ elements of k'_C are the same as the last $(m-X)$ elements of k_C.

Namely:

$$k'_C = (k_1, \cdots, k_{z+X}, \boxed{k_I}, k_{z+X+1}, \cdots, k_{z+m})^T \tag{22}$$

The reason that we construct such k'_C will be explained in Step 4 (tag update).

(2) *C modifies k_r*:

After the insertion, the matrix M is changed as follows:

- In each of the first X rows: a '0' bit is padded in the final position.
- In the inserted row (w_I): v_I is placed in the first z elements, a '1' bit is placed at the $(z+X+1)$-th element counted from the left, and '0' bits are placed elsewhere.
- In each of the last $(m-X)$ rows: a '0' bit is padded in the final position and then, the '1' bit is shipped to the next right position.

$$M = \underbrace{\begin{pmatrix} v_1 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_X & \multicolumn{4}{l}{\underbrace{0 \ \cdots \ 0 \ 1}_{X}} & 0 & \cdots & \cdots & 0 \\ v_{X+1} & \multicolumn{5}{l}{\underbrace{0 \ \cdots \ 0 \ 0 \ 1}_{X+1}} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_m & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}}_{m \times (z+m)}$$

$$M' = \underbrace{\begin{pmatrix} v_1 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_X & \multicolumn{4}{l}{\underbrace{0 \ \cdots \ 0 \ 1}_{X}} & 0 & \cdots & \cdots & \cdots & 0 \\ \boxed{v_I} & \multicolumn{5}{l}{\underbrace{0 \ \cdots \ 0 \ 0 \ 1}_{X+1}} & 0 & \cdots & \cdots & 0 \\ v_{X+1} & \multicolumn{6}{l}{\underbrace{0 \ \cdots \ 0 \ 0 \ 0 \ 1}_{X+2}} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_m & \multicolumn{9}{l}{\underbrace{0 \ \cdots \ \cdots \ \cdots \ \cdots \ \cdots \ \cdots \ 0 \ 1}_{m+1}} \end{pmatrix}}_{(m+1) \times (z+m+1)}$$

We now update k'_r as follows. Let $B_\psi = (b_1, \cdots, b_{z+m})^T$ and $B'_\psi = (b'_1, \cdots, b'_{z+m+1})^T$ where $\psi \in \{1, \cdots, z\}$ be the z basis vectors of M and M', respectively. Given B_ψ, we firstly find

B'_ψ:

- The first $(z+X)$ elements of B'_ψ are the same as the first $(z+X)$ elements of B_ψ.
- The $(z+X+1)$-th elements of B'_ψ is computed as: $b'_{z+X+1} = (-\sum_{\mu=1}^{z} v_{I\mu} b_\mu) \mod q$.
- The last $(m-X)$ elements of B'_ψ are the same as the last $(m-X)$ elements of B_ψ.

In other words:

$$B'_\psi = (b_1, \cdots, b_{z+X}, \boxed{(-\sum_{\mu=1}^{z} v_{I\mu} b_\mu) \mod q}, \\ b_{z+X+1}, \cdots, b_{z+m})^T \tag{23}$$

After having B'_ψ for all $\psi \in \{1, \cdots, z\}$, C computes $k'_p \leftarrow \mathsf{Kg}(B'_1, \cdots, B'_z)$. C finally sends $k'_r = k'_C + k'_p$ to a new server when next repair phase is executed.

(3) *C computes tag for w_I*:

- C computes $t_I = w_I \cdot k'_C$.
- C sends $\{w_I, t_I\}$ to $\mathcal{S}_i$.

(4) *$\mathcal{S}_i$ updates coded blocks and tags*:

Because $v_k \in \mathbb{F}_q^z$, v_k is viewed as a vector of z elements in $\mathbb{F}_q$: $v_k = (v_{k1}, \cdots, v_{kz})$. $w_k \in \mathbb{F}_q^{z+m}$ has the form as Eq. (18). Because a coded block $c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} w_k \in \mathbb{F}_q^{z+m}$, c_{ij} can be also viewed as a vector of $(z+m)$ elements in $\mathbb{F}_q$. Let $c_{ij}[x]$ denote the x-th element of c_{ij} where $x \in \{1, \cdots, z+m\}$ as Eq. (19).

((4).1) <u>$\mathcal{S}_i$ updates its coded blocks:</u>

$$c'_{ij} = \begin{pmatrix} \sum_{k=1}^{m} \alpha_{ijk} v_{k1} + \alpha_{ijI} v_{I1} \\ \vdots \\ \sum_{k=1}^{m} \alpha_{ijk} v_{kz} + \alpha_{ijI} v_{Iz} \\ \alpha_{ij1} \\ \vdots \\ \alpha_{ijX} \\ \boxed{\alpha_{ijI}} \\ \alpha_{ij(X+1)} \\ \vdots \\ \alpha_{ijm} \end{pmatrix}^T = \begin{pmatrix} c_{ij}[1] + \alpha_{ijI} v_{I1} \\ \vdots \\ c_{ij}[z] + \alpha_{ijI} v_{Iz} \\ c_{ij}[z+1] \\ \vdots \\ c_{ij}[z+X] \\ \boxed{\alpha_{ijI}} \\ c_{ij}[z+X+1] \\ \vdots \\ c_{ij}[z+m] \end{pmatrix}^T \tag{24}$$

where $\alpha_{ijI} \overset{rand}{\leftarrow} \mathbb{F}_q$.

((4).2) <u>$\mathcal{S}_i$ updates its tags:</u>

Tags of augmented blocks before the insertion are:
$(t_{w_1}, \cdots, t_{w_X}, t_{w_{(X+1)}}, \cdots, t_{w_m})^T = M \cdot k_C$

$$= \begin{pmatrix} v_1 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_X & \underbrace{0 \ \cdots \ 0 \ \ 1}_{X} & & 0 & \cdots & \cdots & & & 0 \\ v_{X+1} & \underbrace{0 \ \cdots \ 0 \ \ 0}_{X+1} & & 1 & 0 & \cdots & & & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_m & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}$$

$\times(k_1, \cdots, k_{z+1}, \cdots, k_{(z+X)}, k_{(z+X+1)}, \cdots, k_{z+m})^T$

$$= \begin{pmatrix} v_{11}k_1 + \cdots + v_{1z}k_z + k_{z+1} \\ \vdots \\ v_{X1}k_1 + \cdots + v_{Xz}k_z + k_{z+X} \\ v_{(X+1)1}k_1 + \cdots + v_{(X+1)z}k_z + k_{z+X+1} \\ \vdots \\ v_{m1}k_1 + \cdots + v_{mz}k_z + k_{z+m} \end{pmatrix}$$

By constructing k'_C in Step 1, tags of augmented blocks after the insertion are:
$(t'_{w_1}, \cdots, t'_{w_X}, \boxed{t_I}, t'_{w_{(X+1)}}, \cdots, t'_{w_{m+1}})^T = M' \cdot k'_C$

$$= \begin{pmatrix} v_1 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_X & \underbrace{0 \ \cdots \ 0 \ \ 1}_{X} & & 0 & \cdots & \cdots & \cdots & & & 0 \\ \boxed{v_I} & \underbrace{0 \ \cdots \ 0 \ \ 0}_{X+1} & & 1 & 0 & \cdots & \cdots & & & 0 \\ v_{X+1} & \underbrace{0 \ \cdots \ 0 \ \ 0 \ \ 0}_{X+2} & & & 1 & 0 & \cdots & & & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_m & \underbrace{0 \ \cdots \ \cdots \ \cdots \ \cdots \ \cdots \ \cdots \ 0 \ \ 1}_{m+1} \end{pmatrix}$$

$\times(k_1, \cdots, k_z, k_{z+1}, \cdots, k_{(z+X)}, \boxed{k_I}, k_{z+X+1}, \cdots, k_{z+m})^T$

$$= \begin{pmatrix} v_{11}k_1 + \cdots + v_{1z}k_z + k_{z+1} \\ \vdots \\ v_{X1}k_1 + \cdots + v_{Xz}k_z + k_{z+X} \\ v_{I1}k_1 + \cdots + v_{Iz}k_z + k_I \\ v_{(X+1)1}k_1 + \cdots + v_{(X+1)z}k_z + k_{z+X+1} \\ \vdots \\ v_{m1}k_1 + \cdots + v_{mz}k_z + k_{z+m} \end{pmatrix}$$

We can observe that before and after the insertion, the first X tags and the last $(m-X)$ tags are unchanged. Only the new tag t_I (tag of w_I) is inserted. The old tag of c_{ij} is computed as $t_{ij} = \sum_{k=1}^{m} \alpha_{ijk} t_{w_k}$. Thus, we compute the tag for c'_{ij} as follows:

$$\begin{aligned} t_{c'_{ij}} &= \sum_{k=1}^{X} \alpha_{ijk} t_{w_k} + \alpha_{ijI} t_I + \sum_{k=X+1}^{m} \alpha_{ijk} t_{w_k} \\ &= t_{ij} + \alpha_{ijI} t_I \end{aligned} \tag{25}$$

where α_{ijI} is the same as in Eq. (24).

The insertion only needs $O(1)$ for modifying k_C, $O(z)$ for modifying k_r, $O(1)$ for computing tag for w_I and $O(z)$ for *updating a coded block and a tag.*

4.4.3 Deletion

Suppose that C deletes the X-th file block (v_X). Let w_X be the augmented block of v_X.

(1) *C modifies k_C*:

Similar to the insertion, before the deletion, $k_C = (k_1, \cdots, k_{z+m})^T$. After the deletion, C simply removes the $(z+X)$-th element in k_C. Namely,

$$k'_C = (k_1, \cdots, k_{z+X-1}, k_{z+X+1}, \cdots, k_{z+m})^T \tag{26}$$

The reason to construct such k'_C will be explained in Step 3 (tag update).

(2) *C modifies k_r*:

After the deletion, the matrix M is changed as follows:

- In each of the first $(X-1)$ rows, the '0' bit at the final position is removed.
- The X-th row is removed.
- In each of the last $(m-X)$ rows, the '1' bit is shipped to the previous left position and then, the '0' bit at the final position is removed.

$$M = \underbrace{\left(\begin{array}{ll}
v_1 & 1 \quad 0 \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad 0 \\
\vdots & \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
v_{X-1} & \underbrace{0 \quad \cdots \quad 0 \quad 1}_{X-1} \quad 0 \quad \cdots \quad \cdots \quad \cdots \quad 0 \\
\boxed{v_X} & \underbrace{0 \quad \cdots \quad 0 \quad 0 \quad 1}_{X} \quad 0 \quad \cdots \quad \cdots \quad 0 \\
v_{X+1} & \underbrace{0 \quad \cdots \quad 0 \quad 0 \quad 0 \quad 1}_{X+1} \quad 0 \quad \cdots \quad 0 \\
\vdots & \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
v_m & \underbrace{0 \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad 0 \quad 1}_{m}
\end{array}\right)}_{m\times(z+m)}$$

$$M' = \underbrace{\left(\begin{array}{ll}
v_1 & 1 \quad 0 \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad 0 \\
\vdots & \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
v_{X-1} & \underbrace{0 \quad \cdots \quad 0 \quad 1}_{X-1} \quad 0 \quad \cdots \quad \cdots \quad 0 \\
v_{X+1} & \underbrace{0 \quad \cdots \quad 0 \quad 0 \quad 1}_{X} \quad 0 \quad \cdots \quad 0 \\
\vdots & \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
v_m & \underbrace{0 \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad 0 \quad 1}_{m-1}
\end{array}\right)}_{(m-1)\times(z+m-1)}$$

We now update k'_r as follows. Let $B_\psi = (b_1, \cdots, b_{z+m})^T$ and $B'_\psi = (b_1, \cdots, b_{z+m-1})^T$ where $\psi \in \{1, \cdots, z\}$ be the z basis vectors of M and M', respectively. To compute B'_ψ from B_ψ, C simply removes the $(z+X)$-th element of B_ψ. Namely,

$$B'_\psi = (b_1, \cdots, b_{z+X-1}, b_{z+X+1}, \cdots, b_{z+m})^T \tag{27}$$

After having B'_ψ for all $\psi \in \{1, \cdots, z\}$, C computes k'_p as: $k'_p \leftarrow \mathsf{Kg}(B'_1, \cdots, B'_z)$. C finally sends $k'_r = k'_C + k'_p$ to a new server when next repair phase is executed.

(3) *$\mathcal{S}_i$ updates coded blocks and tags*:

Because $v_k \in \mathbb{F}_q^z$, v_k can be viewed as a vector of z elements in $\mathbb{F}_q$: $v_k = (v_{k1}, \cdots, v_{kz})$. $w_k \in \mathbb{F}_q^{z+m}$ has the form as Eq. (18). Because a coded block $c_{ij} = \sum_{k=1}^m \alpha_{ijk} w_k \in \mathbb{F}_q^{z+m}$, c_{ij} can be also viewed as a set of $(z+m)$ elements in $\mathbb{F}_q$. Let $c_{ij}[x]$ denote the x-th element of c_{ij} where $x \in \{1, \cdots, z+m\}$ as Eq. (19).

((3).1) <u>$\mathcal{S}_i$ updates its coded blocks:</u>

$$c'_{ij} = \begin{pmatrix} \sum_{k=1}^m \alpha_{ijk} v_{k1} - \alpha_{ijX} v_{X1} \\ \vdots \\ \sum_{k=1}^m \alpha_{ijk} v_{kz} - \alpha_{ijX} v_{Xz} \\ \alpha_{ij1} \\ \vdots \\ \alpha_{ij(X-1)} \\ \alpha_{ij(X+1)} \\ \vdots \\ \alpha_{ijm} \end{pmatrix}^T = \begin{pmatrix} c_{ij}[1] - \alpha_{ijX} v_{X1} \\ \vdots \\ c_{ij}[z] - \alpha_{ijX} v_{Xz} \\ c_{ij}[z+1] \\ \vdots \\ \alpha_{ij}[z+X-1] \\ \alpha_{ij}[z+X+1] \\ \vdots \\ \alpha_{ij}[z+m] \end{pmatrix}^T \tag{28}$$

where $\alpha_{ijX} = c_{ij}[z+X]$.

((3).2) <u>$\mathcal{S}_i$ updates its tags:</u>

Tags of augmented blocks before the deletion are:
$(t_{w_1}, \cdots, t_{w_{X-1}}, t_{w_X}, t_{w_{(X+1)}}, \cdots, t_{w_m})^T = M \cdot k_C$

$$= \left(\begin{array}{ll}
v_1 & 1 \quad 0 \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad 0 \\
\vdots & \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
v_{X-1} & \underbrace{0 \quad \cdots \quad 0 \quad 1}_{X-1} \quad 0 \quad \cdots \quad \cdots \quad \cdots \quad 0 \\
\boxed{v_X} & \underbrace{0 \quad \cdots \quad 0 \quad 0 \quad 1}_{X} \quad 0 \quad \cdots \quad \cdots \quad 0 \\
v_{X+1} & \underbrace{0 \quad \cdots \quad 0 \quad 0 \quad 0 \quad 1}_{X+1} \quad 0 \quad \cdots \quad 0 \\
\vdots & \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
v_m & 0 \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad 0 \quad 1
\end{array}\right)$$

$$\times (k_1, \cdots, k_z, k_{z+1}, \cdots, k_{z+m})^T$$

$$= \begin{pmatrix} v_{11}k_1 + \cdots + v_{1z}k_z + k_{z+1} \\ \vdots \\ v_{(X-1)1}k_1 + \cdots + v_{(X-1)z}k_z + k_{z+X-1} \\ v_{X1}k_1 + \cdots + v_{Xz}k_z + k_{z+X} \\ v_{(X+1)1}k_1 + \cdots + v_{(X+1)z}k_z + k_{z+X+1} \\ \vdots \\ v_{m1}k_1 + \cdots + v_{mz}k_z + k_{z+m} \end{pmatrix}$$

By constructing k'_C as Step 1, tags of augmented blocks after the deletion are:
$(t'_{w_1}, \cdots, t'_{w_{X-1}}, t'_{w_{(X+1)}}, \cdots, t'_{w_{m+1}})^T = M' \cdot k'_C$

$$= \begin{pmatrix} v_1 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{X-1} & \underbrace{0 \ \cdots \ 0 \ 1}_{X-1} & & & & 0 & \cdots & \cdots & 0 \\ v_{X+1} & \underbrace{0 \ \cdots \ 0 \ 0 \ 1}_{X} & & & & 0 & \cdots & & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_m & \underbrace{0 \ \cdots \ \cdots \ \cdots \ \cdots \ \cdots \ 0}_{m-1} & & & & & & & 1 \end{pmatrix}$$

$$\times(k_1, \cdots, k_z, k_{z+1}, \cdots, k_{z+X-1}, k_{z+X+1}, \cdots, k_{z+m})^T$$

$$= \begin{pmatrix} v_{11}k_1 + \cdots + v_{1z}k_z + k_{z+1} \\ \vdots \\ v_{(X-1)1}k_1 + \cdots + v_{(X-1)z}k_z + k_{z+X-1} \\ v_{(X+1)1}k_1 + \cdots + v_{(X+1)z}k_z + k_{z+X+1} \\ \vdots \\ v_{m1}k_1 + \cdots + v_{mz}k_z + k_{z+m} \end{pmatrix}$$

We observe that before and after the deletion, only X-th tag is removed and the other tags are unchanged. The old tag of c_{ij} is computed as: $t_{ij} = \sum_{k=1}^{m} \alpha_{ijk} t_{w_k}$. Thus, we compute the tag for c'_{ij} as follows:

$$\begin{aligned} t_{c'_{ij}} &= \sum_{k=1}^{X-1} \alpha_{ijk} t_{w_k} + \sum_{k=X+1}^{m} \alpha_{ijk} t_{w_k} \\ &= t_{ij} - \alpha_{ijX} t_X \end{aligned} \tag{29}$$

where α_{ijX} is the same as in Eq. (28).

The deletion only needs $O(1)$ for modifying k_C, $O(z)$ for modifying k_r, $O(z)$ for updating a coded block and tag.

5. Security Analysis

Our scheme is secure from pollution attack and curious attack as follows.

Theorem 3: The D2-POR is secured from the pollution and curious attacks.

Proof:

1. Pollution attack: suppose that $\mathcal{A}$ is a malicious server in a set of l servers used in data repair. $\mathcal{A}$ injects an invalid pair of $(c_{\mathcal{A}}, t_{\mathcal{A}})$ to the new server $\mathcal{S}'_r$. $\mathcal{S}'_r$ will check $(c_{\mathcal{A}}, t_{\mathcal{A}})$ using the key $k_r \in \mathbb{F}_q^{z+m}$. Because $\mathcal{S}'_r$ is assumed to not collude with the other servers, k_r is only known by $\mathcal{S}'_r$. Thus, $\mathcal{A}$ can only pass the verification of $\mathcal{S}'_r$ with a probability $1/q^{z+m}$ via the brute-force search. If q is chosen large enough (e.g, 160 bits), the probability is $1/(2^{160})^{z+m}$, which is negligible.

   Consider that $\mathcal{S}'_r$ itself is a malicious server who will perform a pollution attack in the next epoch. Even though $\mathcal{S}'_r$ holds k_r, $\mathcal{S}'_r$ cannot pass the verification because k_r is a one-time repair key. Another new server will be given a key $k'_r \neq k_r$.
2. Curious attack: the new server is given the key $k_r = k_C + k_p \in \mathbb{F}_q^{z+m}$. Similar to the pollution attack, the probability of the new server to learn k_C is $1/q^{z+m}$ via the brute-force search. This probability is from learning k_C directly or learning k_p and then obtaining k_C by $k_C = k_r - k_p$. If q is chosen large enough (e.g, 160 bits), the probability is $1/(2^{160})^{z+m}$, which is negligible. □

We also show the condition to reconstruct F via the following theorem.

Theorem 4: F can be reconstructed if in any epoch, at least l out of n servers collectively store m coded blocks which are linearly independent combinations of m augmented blocks, and the matrix consisting of the accumulated coefficients has full rank (i.e, the rank is equal to m).

Proof: $\mathcal{S}_i$ stores d coded blocks c_{ij} where $j \in \{1, \cdots, d\}$. c_{ij} is computed from m augmented blocks $w_1, \cdots, w_m$ by $c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} w_k \in \mathbb{F}_q^{z+m}$. Therefore, to reconstruct F, m augmented blocks are viewed as the unknowns that need to be solved. To solve these unknowns, at least m coded blocks are required such that the accumulated coefficient matrix has full rank.

$$\begin{cases} c_{(ij)_1} = \sum_{k=1}^{m} \alpha_{(ijk)_1} \cdot w_k \\ c_{(ij)_2} = \sum_{k=1}^{m} \alpha_{(ijk)_2} \cdot w_k \\ \vdots \\ c_{(ij)_m} = \sum_{k=1}^{m} \alpha_{(ijk)_m} \cdot w_k \end{cases} \tag{30}$$

Let l be the number of servers ($l < n$) which collectively stores these m coded blocks. Because each server stores d coded blocks, $l \geq \lceil \frac{m}{d} \rceil$. □

6. Efficiency Analysis

The feature and efficiency comparison between our scheme and previous schemes (RDC-NC, MD-POR and NC-Audit) is given in Table 2. Because the MD-POR and NC-Audit schemes focus on the public authentication, the system models have one more entity called TPA (Third Party Auditor) who is delegated the task of checking the servers by C. For the fair comparison, we assume that the check task in these schemes is performed by C. Furthermore, the MD-POR deals with multiple clients. Thus, for fair comparison, we set the number of clients in MD-POR is one.

6.1 Repair Capacity

The RDC-NC, NC-Audit, MD-POR and D2-POR schemes follow Theorem 4. The number of healthy servers in each epoch must be $l \geq \lceil \frac{m}{d} \rceil$. The number of corrupted servers is $n - l \leq n - \lceil \frac{m}{d} \rceil$. Thus, the repair capacity is $n - \lceil \frac{m}{d} \rceil$.

6.2 Storage Cost

Client-side. In the RDC-NC, C stores 5 keys in $\mathbb{F}_q^{z+m}$. Thus,

Table 2 The comparison

		RDC-NC [19]	**MD-POR** [25]	**NC-Audit** [26]	**D2-POR**
Feature	Direct repair	No	Yes	Not completed (*)	Yes
	Dynamic operations	No	No	No	Yes
	Symmetric key	Yes	Yes	Yes	Yes
Repair Capacity		$n-\lceil\frac{m}{d}\rceil$	$n-\lceil\frac{m}{d}\rceil$	$n-\lceil\frac{m}{d}\rceil$	$n-\lceil\frac{m}{d}\rceil$
STORAGE					
Client-side		$5(z+m)\log_2 q$	$(z+m+mdn)\log_2 q$	$(z+m)\log_2 q$	$(z+m)\log_2 q$
Server-side		$O(dn(z+m)\log_2 q)$	$O(dn(z+m)\log_2 q)$	$O(dn(z+m)\log_2 q)$	$O(dn(z+m)\log_2 q)$
COMPUTATION					
Encode	Client-side	$O(mnd)$	$O(mnd)$	$O(mnd)$	$O(mnd)$
	Server-side	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Check	Client-side	$O(n)$	$O(n)$	$O(n)$	$O(n)$
	Server-side	$O(dn)$	$O(dn)$	$O(dn)$	$O(dn)$
Repair	Client-side	$O(dl)$	$O(1)$	$O(1)$	$O(1)$
	Server-side	$O(dl)$	$O(dl)$	$O(dl)$	$O(dl)$
Modify	Client-side	N/A	N/A	N/A	$O(z)$
	Server-side	N/A	N/A	N/A	$O(dnz)$
Insert	Client-side	N/A	N/A	N/A	$O(z)$
	Server-side	N/A	N/A	N/A	$O(dnz)$
Delete	Client-side	N/A	N/A	N/A	$O(z)$
	Server-side	N/A	N/A	N/A	$O(dnz)$
COMMUNICATION					
Encode		$O(dn(z+m)\log_2 q)$	$O(dn(z+m)\log_2 q)$	$O(dn(z+m)\log_2 q)$	$O(dn(z+m)\log_2 q)$
Check		$O(n(z+m+1)\log_2 q)$	$O(n(z+m+1)\log_2 q)$	$O(n(z+m+1)\log_2 q)$	$O(n(z+m+1)\log_2 q)$
Repair		$O((l+d)(z+m+1)\log_2 q)$	$O(l(z+m+1)\log_2 q)$	$O(l(z+m+1)\log_2 q)$	$O(l(z+m+1)\log_2 q)$
Modify		N/A	N/A	N/A	$O(((n+1)(z+m)$ $+n)\log_2 q)$
Insert		N/A	N/A	N/A	$O(((n+1)(z+m)$ $+n)\log_2 q)$
Delete		N/A	N/A	N/A	$O((z+m)\log_2 q)$

N/A means not applicable due to the lack of support. (*) In the NC-Audit, the direct repair can lead to the pollution attack because the new server cannot check the provided coded blocks.

the storage cost is $5(z+m)\log_2 q$. In the NC-Audit, C stores a key in $\mathbb{F}_q^{z+m}$ and mnd coefficients in $\mathbb{F}_q$. Thus, the storage cost is $(z+m+mnd)\log_2 q$. In the MD-POR and D2-POR, C stores a key in $\mathbb{F}_q^{z+m}$. Thus, the storage cost is $(z+m)\log_2 q$.

Server-side. In all the schemes, there are n server. Each server stores d coded blocks. Each coded block belongs to $\mathbb{F}_q^{z+m}$. Thus, the storage cost is $O(dn(z+m)\log_2 q)$.

6.3 Computation Cost

Encode. In all the schemes, C needs $O(m)$ to compute m tags for m augmented blocks, and $O(mnd)$ to compute nd coded blocks along with the tags. The complexity on the client-side is thus $O(mnd)$. Meanwhile, the servers only need to receive the coded blocks and tags from C without any computation. The complexity on the server-side is thus $O(1)$.

Check. In all the schemes, C needs $O(1)$ to verify the aggregated coded block and tag of each server. Therefore, the complexity on the client-side is $O(n)$ to verify n servers. Meanwhile, each server needs to combine d coded blocks and d tags to compute the aggregated coded block and aggregated tag, respectively. Therefore, the complexity of n servers is $O(dn)$.

Repair. In the RDC-NC scheme, C needs $O(l)$ to check l pairs of the provided coded block and tag from the healthy servers, and needs $O(dl)$ to compute d pairs of new coded blocks and tags using the linear combinations of l pairs of the provided coded blocks and tags. Therefore, the complexity on the client-side is $O(dl)$. In the MD-POR, NC-Audit and D2-POR schemes, the complexity on the client-side is $O(1)$ because C does not need to do anything due to the direct repair.

In the RDC-NC scheme, each of l servers combines its d coded blocks and d tags to compute the aggregated coded block and aggregated tag, respectively. Therefore, the complexity on the server-side is $O(dl)$. In the MD-POR, NC-Audit and D2-POR schemes, l healthy servers perform as in the RDC-NC ($O(dl)$), and the new server performs the task of C as in the RDC-NC ($O(dl)$). Therefore, the complexity on the server-side is $O(dl)$.

Modification. In the D2-POR, C only needs $O(z)$ to recompute k_r (Step 1) and $O(1)$ to compute new tag for the modified augmented block (Step 2). Therefore, the complexity on client-side is $O(z)$. Meanwhile, each server needs $O(dz)$ to update coded blocks and tags (Step 3). Therefore, the complexity of n servers is $O(dnz)$.

Insertion. In the D2-POR, C only needs $O(1)$ to recompute k_C (Step 1), $O(z)$ to recompute k_r (Step 2) and $O(1)$ to compute tag of the inserted augmented block (Step 3). Therefore, the complexity on client-side is $O(z)$. Meanwhile each server needs $O(dz)$ to update coded blocks and tags (Step 4).

Therefore, the complexity of n servers is $O(dnz)$.

Deletion. In the D2-POR, C only needs $O(1)$ to recompute k_C (Step 1) and $O(z)$ to recompute k_r (Step 2). Thus, the complexity on client-side is $O(z)$. Meanwhile, each server needs $O(dz)$ to update the coded blocks and tags (Step 3). Thus, the complexity of n servers is $O(dnz)$.

6.4 Communication

Encode. In all the schemes, C computes dn coded blocks and sends d coded blocks to each of n servers. The size of a coded block is $((z+m)\log_2 q)$. Thus, the communication cost is $O(dn(z+m)\log_2 q)$.

Check. In all the schemes, during the check phase, each of n servers sends its aggregated coded block and its aggregated tag to C. The size of a coded block and a tag is $((z+m)\log_2 q)$ and $\log_2 q$, respectively. Thus, the communication cost is $O(n(z+m+1)\log_2 q)$.

Repair. In the RDC-NC, each of l healthy servers sends to C an aggregated coded block whose size is $((z+m)\log_2 q)$ and an aggregated tag whose size is $\log_2 q$ (it takes $O(l(z+m+1)\log_2 q)$). After computing d new coded blocks and d new tags, C sends them to new server ($O(d(z+m+1)\log_2 q)$). Therefore, the communication cost is $O((l+d)(z+m+1)\log_2 q)$. In the NC-Audit, MD-POR and D2-POR, each of l healthy servers sends directly the aggregated coded block and aggregated tag to the new server. Thus, the communication cost is $O(l(z+m+1)\log_2 q)$.

Modify. In the D2-POR, C sends the updated $k_r \in \mathbb{F}_q^{z+m}$ to new server in Step 1 ($O((z+m)\log_2 q)$), and send ($w'_X \in \mathbb{F}_q^{z+m}, t'_X \in \mathbb{F}_q$) to each $\mathcal{S}_i$ where $i \in \{1,\cdots,n\}$ in Step 2 ($O(n(z+m+1)\log_2 q)$). Thus, the communication cost is $O((z+m)\log_2 q + n(z+m+1)\log_2 q) = O(((n+1)(z+m)+n)\log_2 q)$.

Insert. In the D2-POR, C sends the updated $k_r \in \mathbb{F}_q^{z+m}$ to new server in Step 2 ($O((z+m)\log_2 q)$), and send ($w_I \in \mathbb{F}_q^{z+m}, t_I \in \mathbb{F}_q$) to each $\mathcal{S}_i$ where $i \in \{1,\cdots,n\}$ in Step 3 ($O(n(z+m+1)\log_2 q)$). Thus, the communication cost is $O((z+m)\log_2 q + n(z+m+1)\log_2 q) = O(((n+1)(z+m)+n)\log_2 q)$.

Delete. In the D2-POR, C only needs to send the updated $k_r \in \mathbb{F}_q^{z+m}$ to new server in Step 2. Thus, the communication cost is $O((z+m)\log_2 q)$.

7. Numeric Example

Suppose $m=3$, $z=1$, $q=7$. The file blocks are:

$$\begin{cases} v_1 = 3 \in \mathbb{F}_7^1 \\ v_2 = 4 \in \mathbb{F}_7^1 \\ v_3 = 2 \in \mathbb{F}_7^1 \end{cases} \tag{31}$$

The augmented blocks are:

$$\begin{cases} w_1 = (v_1,1,0,0) = (3,1,0,0) \in \mathbb{F}_7^4 \\ w_2 = (v_2,0,1,0) = (4,0,1,0) \in \mathbb{F}_7^4 \\ w_3 = (v_3,0,0,1) = (2,0,0,1) \in \mathbb{F}_7^4 \end{cases} \tag{32}$$

7.1 Generating Keys

Key for client. $k_C \overset{rand}{\leftarrow} \mathbb{F}_7^4$. Suppose:

$$k_C = (k_1,k_2,k_3,k_4)^T = (1,2,3,4)^T \in \mathbb{F}_7^4 \tag{33}$$

Key for new server. $k_r = k_C + k_p \in \mathbb{F}_7^4$. k_p is generated such that $w_k k_p = 0$ for all $k \in \{1,\cdots,m\}$ as follows:

- Construct a matrix M consisting of all augmented blocks:

$$M = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{pmatrix} \tag{34}$$

- Transform M by Gaussian row echelon in $\mathbb{F}_7$ to obtain M' as follows:

$$M' = \begin{pmatrix} \boxed{1} & 5 & 0 & 0 \\ 0 & \boxed{1} & 1 & 0 \\ 0 & 0 & \boxed{1} & 5 \end{pmatrix} \tag{35}$$

- Let $x = (x_1,x_2,x_3,x_4)^T$. Solve $M'x = 0$, we have:

$$\begin{cases} x_1 + 5x_2 = 0 \\ x_2 + x_3 = 0 \\ x_3 + 5x_4 = 0 \end{cases} \tag{36}$$

- From M', determine free variables = $\{x_4\}$ and pivot variables = $\{x_1,x_2,x_3\}$. Equation (36) yields:

$$(x_1,x_2,x_3,x_4)^T = x_4(3,5,2,1)^T \tag{37}$$

- Thus, the basis vector of M is:

$$B = (b_1,b_2,b_3,b_4)^T = (3,5,2,1)^T \tag{38}$$

- Generate randomly $r=3$ in $\mathbb{F}_7$.
- Compute k_p as:

$$k_p = rB = 3(3,5,2,1)^T \mod 7 = (2,1,6,3)^T \tag{39}$$

It is clear that $w_1 k_p \mod 7 = w_2 k_p \mod 7 = w_3 k_p \mod 7 = 0$**.**

- Finally, k_r is computed as:

$$\begin{aligned} k_r &= k_C + k_p \\ &= (1,2,3,4)^T + (2,1,6,3)^T \mod 7 \\ &= (3,3,2,0)^T \end{aligned} \tag{40}$$

7.2 Dynamic Operations

7.2.1 Modification

Suppose $v_2 = 4$ is modified to $v'_2 = 5$. Matrix M is changed as follows:

$$M = \begin{pmatrix} 3 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{pmatrix} \rightarrow M' = \begin{pmatrix} 3 & 1 & 0 & 0 \\ \boxed{5} & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{pmatrix} \tag{41}$$

(1) *C recomputes k_r:*

- Recompute basis vector:

$$B' = \begin{pmatrix} b_1 \\ b_2 \\ -b_1 v'_2 \mod 7 \\ b_4 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ 6 \\ 1 \end{pmatrix} \tag{42}$$

- Generate randomly $r' = 2 \in \mathbb{F}_7$.
- Recompute k_p:

$$\begin{aligned} k'_p &= r'B' = 2(3,5,6,1)^T \mod 7 \\ &= (6,3,5,2)^T \end{aligned} \tag{43}$$

It is clear that $k'_p w_1 \mod 7 = k'_p w'_2 \mod 7 = k'_p w_3 \mod 7 = 0$**.**

- Recompute k_r:

$$\begin{aligned} k'_r &= k_C + k'_p \\ &= (1,2,3,4)^T + (6,3,5,2)^T \mod 7 \\ &= (0,5,1,6)^T \end{aligned} \tag{44}$$

(2) *C computes tag for w'_2:*

- C computes:

$$\begin{aligned} t'_2 &= w'_2 k_C = (5,0,1,0)(1,2,3,4)^T \mod 7 \\ &= 1 \end{aligned} \tag{45}$$

- C sends $\{w'_2, t'_2\}$ to $\mathcal{S}_i$

(3) *$\mathcal{S}_i$ recomputes coded blocks and tags:*

Let $\{c_{ij}[1], \cdots, c_{ij}[4]\}$ denote the elements of c_{ij}:

$$c_{ij} = \begin{pmatrix} \sum_{k=1}^{m} \alpha_{ijk} w_k \\ \alpha_{ij1} \\ \alpha_{ij2} \\ \alpha_{ij3} \end{pmatrix}^T = \begin{pmatrix} c_{ij}[1] \\ c_{ij}[2] \\ c_{ij}[3] \\ c_{ij}[4] \end{pmatrix}^T \tag{46}$$

$\mathcal{S}_i$ update coded blocks:

$$c_{ij} = \begin{pmatrix} c_{ij}[1] + \alpha_{ij2}(w'_2 - w_2) \\ c_{ij}[2] \\ c_{ij}[3] \\ c_{ij}[4] \end{pmatrix}^T \tag{47}$$

$\mathcal{S}_i$ updates tags:

$$t'_{ij} = t_{ij} + \alpha_{ij2}(t'_2 - t_2) \tag{48}$$

where $\alpha_{ij2} = c_{ij}[3]$.

7.2.2 Insertion

Suppose $v_I = 1$ is inserted after v_2. The matrix M is changed as follows:

$$M = \begin{pmatrix} 3 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{pmatrix} \rightarrow M' = \begin{pmatrix} 3 & 1 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{49}$$

(1) *C recomputes k_C:*

$$k'_C = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ k_I \overset{rand}{\leftarrow} \mathbb{F}_q \\ k_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 5 \\ 4 \end{pmatrix} \tag{50}$$

(2) *C recomputes k_r:*

- Recompute basis vector:

$$B' = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ -v_I b_1 \mod 7 \\ b_4 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ 2 \\ 4 \\ 1 \end{pmatrix} \tag{51}$$

- Generate randomly $r' = 4 \in \mathbb{F}_7$.
- Recompute k_p:

$$\begin{aligned} k'_p &= r'B' = 4(3,5,2,4,1)^T \mod 7 \\ &= (5,6,1,2,4)^T \end{aligned} \tag{52}$$

It is clear that $w_1 k'_p \mod 7 = w_2 k'_p \mod 7 = w_I k'_p \mod 7 = w_3 k'_p \mod 7 = 0$**.**

- Recompute k_r:

$$\begin{aligned} k'_r &= k'_C + k'_p \\ &= (1,2,3,5,4)^T + (5,6,1,2,4)^T \mod 7 \\ &= (6,1,4,0,1)^T \end{aligned} \tag{53}$$

(3) *C computes tag for w'_2:*

- C computes:

$$t_I = w_I k'_C = (1,0,0,1,0)(1,2,3,5,4)^T = 6 \tag{54}$$

- C sends $\{w_I, t_I\}$ to $\mathcal{S}_i$.

(4) *$\mathcal{S}_i$ recomputes coded blocks and tags:*

Let $\{c_{ij}[1], \cdots, c_{ij}[4]\}$ denote the elements of c_{ij}:

$$c_{ij} = \begin{pmatrix} \sum_{k=1}^{m} \alpha_{ijk} w_k \\ \alpha_{ij1} \\ \alpha_{ij2} \\ \alpha_{ij3} \end{pmatrix}^T = \begin{pmatrix} c_{ij}[1] \\ c_{ij}[2] \\ c_{ij}[3] \\ c_{ij}[4] \end{pmatrix}^T \tag{55}$$

$\mathcal{S}_i$ updates coded blocks:

$$c'_{ij} = \begin{pmatrix} c_{ij}[1] + \alpha_{ijI} w_I \\ c_{ij}[2] \\ c_{ij}[3] \\ \alpha_{ijI} \overset{rand}{\leftarrow} \mathbb{F}_q \\ c_{ij}[4] \end{pmatrix}^T \quad (56)$$

$\mathcal{S}_i$ updates tags:

$$t'_{ij} = t_{ij} + \alpha_{ijI} t_I \quad (57)$$

7.2.3 Deletion

Suppose $v_2 = 4$ is deleted. The matrix M is changed as follows:

$$M = \begin{pmatrix} 3 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{pmatrix} \rightarrow M' = \begin{pmatrix} 3 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix} \quad (58)$$

(1) *C recomputes* k_C:

$$k'_C = (k_1, k_2, k_4)^T = (1, 2, 4)^T \quad (59)$$

(2) *C recomputes* k_r:

- Recompute basis vector:

$$B' = (b_1, b_2, b_4)^T = (3, 5, 1)^T \quad (60)$$

- Generate randomly $r' = 3 \in \mathbb{F}_7$.
- Recompute k_p:

$$k'_p = r'B' = 3(3, 5, 1)^T \mod 7 = (2, 1, 3)^T \quad (61)$$

It is clear that $w_1 k'_p \mod 7 = w_3 k'_p \mod 7 = 0$.
- Recompute k_r:

$$\begin{aligned} k'_r &= k'_C + k'_p = (1, 2, 4)^T + (2, 1, 3)^T \mod 7 \\ &= (3, 3, 0)^T \end{aligned} \quad (62)$$

(3) *$\mathcal{S}_i$ recomputes coded blocks and tags:*

Let $\{c_{ij}[1], \cdots, c_{ij}[4]\}$ denote the elements of c_{ij}:

$$c_{ij} = \begin{pmatrix} \sum_{k=1}^{m} \alpha_{ijk} w_k \\ \alpha_{ij1} \\ \alpha_{ij2} \\ \alpha_{ij3} \end{pmatrix}^T = \begin{pmatrix} c_{ij}[1] \\ c_{ij}[2] \\ c_{ij}[3] \\ c_{ij}[4] \end{pmatrix}^T \quad (63)$$

$\mathcal{S}_i$ updates coded blocks:

$$c'_{ij} = \begin{pmatrix} c_{ij}[1] - \alpha_{ij2} w_2 \\ c_{ij}[2] \\ c_{ij}[3] \\ c_{ij}[4] \end{pmatrix}^T \quad (64)$$

$\mathcal{S}_i$ updates tags:

$$t'_{ij} = t_{ij} - \alpha_{ij2} t_2 \quad (65)$$

where $\alpha_{ij2} = c_{ij}[3]$.

8. Conclusion

In this paper, we have proposed a network coding-based POR scheme, name D2-POR, to support the direct repair and the dynamic operations in a symmetric key setting. The idea is based on the InterMac technique which can generate a key such that the key is orthogonal to the augmented blocks. The security analysis is showed to prevent the pollution attack and curious attack. The efficiency analysis is given based on complexity theory to compare with the previous scheme.

References

[1] A. Juels and B.S. Kaliski, Jr., "Pors: proofs of retrievability for large files," In: Proc. of 14th conf. on Computer and communications security - CCS'07, pp.584–597, 2007.
[2] H. Shacham and B. Waters, "Compact Proofs of Retrievability," In: Proc. of 14th Conf. on the Theory and Application of Cryptology and Information Security - ASIACRYPT'08, pp.90–107, 2008.
[3] K.D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: theory and implementation," In: Proc. of ACM 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," In: Prof. of conf. on Measurement and modeling of computer systems - SIGMETRICS'00, pp.34–43, 2000.
[5] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession," In: Proc. of 28th conf. on Distributed Computing Systems - ICDCS'08, pp.411–420, 2008.
[6] Z. Zhang, Q. Lian, S. Lin, W. Chen, Y. Chen, and C. Jin, "Bitvault: A highly reliable distributed data retention platform," In: SIGOPS Operating Systems Review, vol.41, no.2, pp.27–36, 2007.
[7] M. Aguilera, R. Janakiraman, and L. Xu, "Efficient fault-tolerant distributed storage using erasure codes," Tech. Rep., Washington University in St. Louis, 2004.
[8] K.D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud Storage," In: Proc. of 16th conf. on Computer and communications security - CCS'09, pp.187–198, 2009.
[9] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of Retrievability via Hardness Amplification," In: Proc. of 6th Theory of Cryptography Conf. on Theory of Cryptography - TCC'09, pp.109–127, 2009.
[10] J. Hendricks, G.R. Ganger, and M.K. Reiter, "Verifying distributed erasure-coded data," In: Proc. of 26th Principles of Distributed Computing Symposium, pp.139–146, 2007.
[11] R. Ahlswede, N. Cai, S.-Y.R. Li, and R.W. Yeung, "Network information flow," In: IEEE Trans. on Information Theory, vol.46, no.4, pp.1204–1216, 2000.
[12] S.-Y.R. Li, R.W. Yeung, and N. Cai, "Linear Network Coding," In: IEEE Trans. on Information Theory, vol.49, no.2, pp.371–381, 2003.
[13] R. Koetter and M. Medard, "An Algebraic Approach to Network Coding," In: IEEE/ACM Trans. on Networking (TON), vol.11, no.5, pp.782–795, 2003.
[14] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-based integrity for network coding," In: Proc. of the Applied Cryptography and Network Security, vol.5536, pp.292–305, 2009.
[15] C. Cheng and T. Jiang, "An efficient homomorphic MAC with small key size for authentication in network coding," In: IEEE Trans. on Computers, vol.62, no.10, pp.2096–2100, 2013.

[16] C. Cheng, T. Jiang, and Q. Zhang, "TESLA-based Homomorphic MAC for Authentication in P2P System for Live Streaming with Network Coding," In: IEEE Journal on Selected Areas in Communications, vol.31, no.9, pp.291–298, 2013.
[17] A.G. Dimakis, P.B. Godfrey, Y. Wu, M.J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," In: IEEE Trans. Information Theory, vol.56, no.9, pp.4539–4551, 2010.
[18] J. Li, S. Yang, X. Wang, X. Xue, and B. Li, "Tree-structured Data Regeneration in Distributed Storage Systems with Network Coding," In: Proc. of 29th IEEE Information commun. Conf., pp.2892–2900, 2000.
[19] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-based Distributed Storage Systems," In: Proc. of ACM workshop on cloud computing security, pp.31–42, 2010.
[20] 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, vol.63, no.1, pp.31–44, 2014.
[21] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious ram," In: Proc. of Conf. on Theory and Applications of Cryptographic Techniques - EUROCRYPT 2013.
[22] S. Elaine, S. Emil, and P. Charalampos, "Practical dynamic proofs of retrievability," In: Proc. of SIGSAC conf. on Computer & communications security - CCS'13, pp.325–336, 2013.
[23] B. Chen and R. Curtmola, "Robust dynamic remote data checking for public clouds," In: Proc. of Conf. on Computer and communications security - CCS'12, pp.1043–1045, 2012.
[24] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling Public Auditability and Data Dynamics for Storage Security in cloud Computing," In: IEEE Trans. parallel and distributed system, vol.22, no.5, pp.847–859, 2011.
[25] K. Omote and T.P. Thao, "MD-POR: Multi-source and Direct Repair for Network Coding-based Proof of Retrievability," In: Journal of Distributed Sensor Networks (IJDSN), Article ID:586720, vol.9198, pp.713–730, 2015.
[26] A. Le and A. Markopoulou, "NC-Audit: Auditing for network coding storage," In: Proc. of IEEE Int. Symposium on Network Coding - NetCod'12, pp.155–160, 2012.
[27] A. Le A and A. Markopoulou, "On detecting pollution attacks in inter-session network coding," In: Proc. of 31st IEEE conf. on Computer Communications - INFOCOM'12, pp.343–351, 2012.
[28] K. Omote and T. Thao, "DD-POR: Dynamic Operations and Direct Repair in Network Coding-based Proof of Retrievability," In: Proc. of 21st Annual International Computing and Combinatorics Conference - COCOON'15, vol.9198, pp.713–730, 2015.

Kazumasa Omote received his M.S. and Ph.D. degrees in information science from Japan Advanced Institute of Science and Technology (JAIST) in 1999 and 2002, respectively. He joined Fujitsu Laboratories, LTD from 2002 to 2008 and engaged in research and development for network security. He has been a research assistant professor at the Japan Advanced Institute of Science and Technology (JAIST) since 2008. His research interests include applied cryptography and network security. He is a member of the IPS of Japan.

Phuong-Thao Tran received her M.S. degree in information science from Japan Advanced Institute of Science and Technology (JAIST) in 2012. She is now the third-year PhD candidate in information science from Japan Advanced Institute of Science and Technology (JAIST).