IEICE TRANSACTIONS

on Information and Systems

VOL. E98-D NO. 8
AUGUST 2015

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

ND-POR: A POR Based on Network Coding and Dispersal Coding*

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

SUMMARY Nowadays, many individuals and organizations tend to outsource their data to a cloud storage for reducing the burden of data storage and maintenance. However, a cloud provider may be untrustworthy. The cloud thus leads to a numerous security challenges: data availability, data integrity, and data confidentiality. In this paper, we focus on data availability and data integrity because they are the prerequisites of the existence of a cloud system. The approach of this paper is the network coding-based Proof of Retrievability (POR) scheme which allows a client to check whether his/her data stored on the cloud servers are intact. Although many existing network coding-based PORs have been proposed, most of them still incur high costs in data check and data repair, and cannot prevent the small corruption attack which is a common attack in the POR scheme. This paper proposes a new network coding-based POR using the dispersal coding technique, named the ND-POR (Network coding - Dispersal coding POR) to improve the efficiency in data check and data repair and to protect against the small corruption attack.

key words: *data integrity, data availability, proof of retrievability, network coding, dispersal coding, cloud storage*

1. Introduction

1.1 Background

Since amount of data is increasing exponentially, data storage and management become increasingly troublesome tasks for the data owners. To reduce the burdens for the data owners, the concept of remote storage known as *cloud* has been proposed. A cloud is considered as a service through which the clients can use to publish, access, manage and share their data remotely and easily from anywhere via the Internet. However, the shortcoming of this system is that a cloud storage provider could not be necessarily trusted. The cloud system thus introduces three security challenges: data availability, data integrity and data confidentiality. Because data availability and data integrity are the pre-conditions for the existence of a system, they are more important than data confidentiality. Therefore, this paper focuses on data availability and data integrity rather than data confidentiality.

Proof of Retrievability. To assist the client in checking whether cloud servers satisfy data availability and data integrity, researchers proposed Provable Data Possession (PDP) [1], [2] and Proof of Retrievability (POR) [3]–[5] which are challenge-response protocols between a client and cloud servers. Both protocols support data check. However, only the POR can ensure that the data are always retrievable. Thus, the POR is considered to be a stronger tool. The POR protocol consists of four phases: keygen, encode, check and repair.

Approaches. Based on the POR protocol, the following two approaches are commonly used. In the first approach, the data are only stored on a single server [3], [4]. The client periodically checks the data and can thus detect data corruption. However, this approach does not allow data repair when a corruption is detected. In the second approach, the client stores data redundantly on multiple servers. Many papers focus on this approach, e.g., [5]–[14]. When a corrupted server is detected, the client can use the remaining healthy servers to repair the data stored on the corrupted server. In this approach, there are three common techniques: *replication*, *erasure coding* and *network coding*.

- *Replication.* Curtmola et al. [9] proposed this technique which allows the client to store a file replica on each server. When a corruption is detected, the client uses one of healthy replicas to repair the corruption. The drawback of this technique, however, is high storage cost because the client must store a whole file on each server.

- *Erasure coding.* Because replication has high storage cost, erasure coding is applied to outsourced data [10] to provide space-optimal data redundancy. In this technique, each server stores file blocks (portions of the file) instead of file replica (copy of the whole file) like replication. The size of the file blocks stored on each server is less than the size of the whole file. Thus, erasure coding can reduce storage cost of replication. However, the drawback of this technique is that to repair a corrupted data, the client must reconstruct the entire file before generating new coded blocks. Therefore, this technique increases computation cost and communication cost in data repair.

- *Network coding.* To improve the efficiency in data repair, researchers apply network coding technique to such outsourced data [11]. Unlike erasure coding, the

Manuscript received January 16, 2015.
Manuscript revised April 21, 2015.
Manuscript publicized May 15, 2015.

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

*The preliminary version of this paper was presented at AINA'14 [32]. This study is partly supported by Grant-in-Aid for Young Scientists (B) (25730083) and CREST, JST.

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

DOI: 10.1587/transinf.2015EDP7011

client does not need to reconstruct the entire file before generating new coded blocks. Instead, the coded blocks which are retrieved from the healthy servers are used to generate new coded blocks. Therefore, this paper focuses on the network coding technique.

The data cannot be checked without embedded information, i.e, Message Authentication Code (MAC) or digital signature. The MAC is sometimes called *tag*. The MAC is used in the symmetric key setting while the digital signature is used in the asymmetric key setting. Because this paper is based on the symmetric key setting, the MAC is used in the proposed scheme.

Network Coding-based POR. Based on the POR, many schemes have been proposed, for instance, [8], [9] using replica, [5]–[7] using erasure coding. However, we are only aware of a few network coding-based POR schemes. Dimakis et al. [12] was the first to apply the network coding to achieve a remarkable reduction in the communication overhead of the repair component. Li et al. [13] 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. Chen et al. [14] presented the Remote Data Checking for Network Coding-based distributed storage system (RDC-NC) scheme which provides a decent solution for efficient data repair by recoding encoded blocks on the healthy servers during the repair procedure. Le et al. [15] introduced the NC-Audit scheme for efficient check and repair using a new homomorphic MAC technique called SpaceMac. Recently, Chen et al. [16] have proposed the NC-Cloud scheme to improve the cost-effectiveness of repair using the functional minimum-storage regenerating (FMSR) codes, which lightens the encoding requirement of storage nodes during repair.

In these network coding-based POR schemes, the most notable scheme is the RDC-NC scheme [14]. It, unlike the other previous schemes, not only focuses on the efficiency, but also considers how to prevent the three common attacks of the POR: *replay attack*, *pollution attack* and *large corruption attack*. However, the RDC-NC scheme has some shortcomings: (i) the corruption check is still inefficient because only one server can be checked per challenge and (ii) it cannot prevent another common attack of the POR: *small corruption attack*. The small corruption attack is defined in [6], [17]–[19]. In this attack, the adversary tries to corrupt the data with a small data unit to hide data loss incidents. Protecting against the small corruption attack protects the data itself, not just the storage resource. Modifying a single bit may destroy an encrypted file or invalidate authentication information. The difference between the large and small corruption attacks is that the small corruption attack corrupts at most t-fraction of the file while the large corruption attack corrupts more than t-fraction of the file, where t is a parameter. These are described more details in the adversarial model (Sect. 3).

To address the small corruption attack, the common solution is to use the Error-Correcting Code (ECC) [20], which allows the data to be checked for errors and corrected even one bit on the fly. The ECC has several types, i.e., Hamming code, Golay code, Reed-Muller code, Reed-Solomon code, etc. However, this paper uses the Reed-Solomon code because the Universal Hash Function can be constructed using the Reed-Solomon code. Bowers et al. [6] then proposed the *dispersal coding* using the Reed-Solomon code in order to prevent the small corruption attack and to ensure the file integrity with high probability. However, [6] uses the erasure coding instead of the network coding.

1.2 Contribution

This paper proposes a new POR scheme using the network coding and the dispersal coding, called ND-POR. To the best of our knowledge, the ND-POR scheme is the first POR to apply both the dispersal coding and the network coding. The contribution of the ND-POR scheme is described as follows:

Security. The ND-POR scheme, unlike the RDC-NC scheme, can prevent the small corruption attack.

Efficiency.

- The RDC-NC scheme allows the client to check one server for each challenge. Meanwhile, the ND-POR scheme allows the client to check all servers simultaneously for each challenge.
- In the RDC-NC scheme, the number of MACs is $n\alpha s$ where n denotes the number of servers, α denotes the number of coded blocks stored on a server and s denotes the number of segments in a coded block. In the ND-POR scheme, the number of MACs is only $l\alpha$ where l denotes some servers out of n servers ($l < n$) and is far less than the dominant parameter s.
- In data repair, the RDC-NC scheme uses the network coding to repair the corruptions. Meanwhile, the ND-POR scheme performs two phases: if the number of corruptions is smaller than the ECC boundary, the ECC is used to repair the corruptions; otherwise the network coding is used to repair the corruptions. Thus, the corruptions are repaired with an overwhelming probability. Furthermore, the ECC uses the parity information on the server itself to repair without the other healthy servers as the network coding.

The dispersal coding is constructed based on UMAC (MAC obtained from Universal Hash Function) which is closely related to the network coding-based schemes [21], [22]. Hence, the network coding and the dispersal coding can be suitably combined together in the ND-POR scheme.

1.3 Organization

The background of the Proof of Retrievability, the network

coding and the dispersal coding are described in Sect. 2. The adversarial model is given in Sect. 3. The ND-POR scheme is proposed in Sect. 4. The security and efficiency analyses are discussed in Sect. 6. The conclusion and future work are drawn in Sect. 7.

2. Preliminaries

2.1 System Model

The ND-POR scheme has two entities. The first entity is the client who can be individuals or organizations. The client outsources his/her data to a cloud storage and relies on the cloud storage for data storage and maintenance. The second entity is the cloud servers which are managed by a cloud provider. The cloud servers store the data of the clients and have responsibility to prove to the client that the stored data are always available and intact.

2.2 Proof of Retrievability

To check whether the availability and integrity of the data stored on the cloud servers are satisfied, researchers proposed Proof of Retrievability (POR) [3]–[5] which is a challenge-response protocol between a client (verifier) and a server (prover). A POR protocol has the following phases:

- keygen(1^λ): Given a security parameter λ, the client generates a secret key (sk) and a public key (pk). For symmetric key setting, pk is set to be null.
- encode(sk, F): The client encodes an original file (F) to an encoded file (F'), then stores F' on the server.
- check(sk): The client uses sk to generate a challenge (c) and sends c to the server. The server computes a response (r) and sends r back to the client. The client then verifies r to determine whether F is available and intact.
- repair(): If a corruption is detected in the check phase, the client will execute this phase to repair the corrupted data. The technique of this phase depends on the each specific scheme, e.g., replication, erasure coding or network coding.

2.3 Network Coding

The network coding [11], [21] offers a good trade-off in terms of redundancy, reliability, and repair bandwidth. The network coding is firstly proposed in the network scenario, and is then applied in the distributed storage system scenario.

Fundamental Concept. In the network scenario, suppose that a source node wants to send a file F to a receiver node via the network. The source node firstly divides F into m blocks: $F = v_1 \| \cdots \| v_m$. $v_k \in \mathbb{F}_p^z$ where $k \in \{1, \cdots, m\}$ and $\mathbb{F}_p^z$ denotes a z-dimensional finite field of a prime order p. The source node then augments v_k with a vector of length m which consists of a single '1' in the k-th position and '0' elsewhere. Let $\{b_1, \cdots, b_m\}$ denote the augmented blocks. b_k has the following form:

$$b_k = (v_k, \overbrace{\underbrace{0, \cdots, 0, 1}_{k}, 0, \cdots, 0}^{m}) \in \mathbb{F}_p^{z+m} \quad (1)$$

The source node sends $\{b_1, \cdots, b_m\}$ as packets to the network. Suppose that an intermediate node in the network receives θ packets $\{b_{i_1}, \cdots, b_{i_\theta}\}$. The intermediate node generates θ coefficients $\alpha_1, \cdots, \alpha_\theta \in \mathbb{F}_p$ and linearly combines the received packets and transmits the resulting linear combination to the adjacent nodes. Therefore, each packet carries m accumulated coefficients which produce that packet as a linear combination of all m augmented blocks. The receiver node can retrieve the augmented blocks from any set of m combinations. If $y \in \mathbb{F}_p^{z+m}$ is a linear combination of $b_1, \cdots, b_m \in \mathbb{F}_p^{z+m}$, then the file blocks $v_1, \cdots, v_m$ can be calculated from the first coordinate of y using the coefficients that contained in the last m coordinates of y.

Application in Distributed Storage System. In the network scenario, there are multiple entities: source node, intermediate nodes and receiver node. However, when the network coding is applied in the distributed storage system scenario, there are only two entities: a client and cloud servers. Let $w = z+m$ where z and m are introduced in the fundamental concept. From the original file $F = \{v_1, \cdots, v_m\}$ ($v_k \in \mathbb{F}_p^z$), the client firstly creates m augmented blocks $\{b_1, \cdots, b_m\} \in \mathbb{F}_p^w$. The client then chooses m coefficients $\{\alpha_1, \cdots, \alpha_m\} \in \mathbb{F}_p$ and linearly combines m augmented blocks to create the coded blocks as $c = \sum_{k=1}^m \alpha_k \cdot b_k \in \mathbb{F}_p^w$. The client stores the coded blocks on the servers. $\{\alpha_1, \cdots, \alpha_m\}$ are chosen such that the matrix which consists of all the coefficients of the coded blocks has full rank. Koetter et al. [23] proved that if the prime p is chosen large enough and the coefficients are chosen randomly, the matrix will have full rank with a high probability. When a corruption is detected, the client retrieves the coded blocks from the healthy servers and linearly combines them to regenerate new coded blocks. For example, in Fig. 1, from the augmented blocks $\{b_1, b_2, b_3\}$, the client chooses the coefficients to compute six coded blocks. The client stores two coded blocks on each of the servers $\{S_1, S_2, S_3\}$. Suppose that S_1 is corrupted, the client requests S_2 and S_3 to compute the aggregated coded blocks by themselves using the linear combinations. The client finally mixes the aggregated coded blocks to obtain two new coded blocks for the new server.

2.4 Dispersal Coding

To prevent the small corruption attack and to allow the client to repair the data with a high probability, the dispersal coding is proposed [6] with a minimal additional storage over-

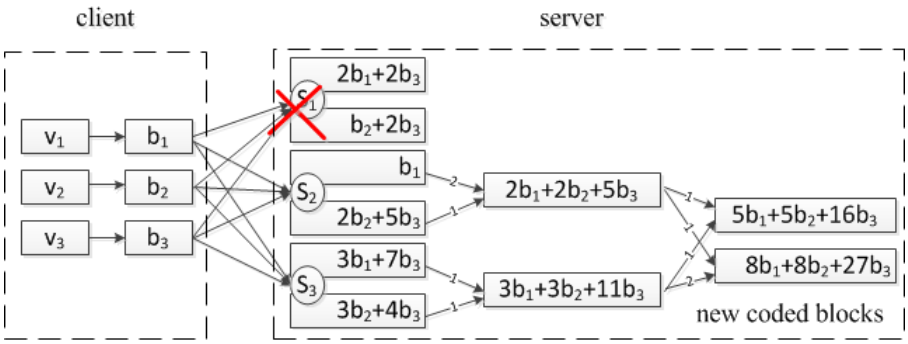

Fig. 1 An example of the network coding

head. The dispersal coding is constructed into a single primitive called *codeword* using the following building blocks.

Universal Hash Function (UHF)**.** A UHF [24] is a function h: $\mathcal{K} \times \mathcal{I}^l \rightarrow \mathcal{I}$ where $\mathcal{I}$ denotes a field with operations $(+, \times)$. This UHF compresses a message $m \in \mathcal{I}^l$ into a compact digest based on a key $\kappa \in \mathcal{K}$ such that the hash of two different messages is different with an overwhelming probability over keys. A common UHF is almost XOR universal (AXU) which satisfies:

- h is an ϵ-UHF family if $\forall x \neq y \in \mathcal{I}^l : \mathsf{Pr}_{\kappa \leftarrow \mathcal{K}}[h_\kappa(x) = h_\kappa(y)] \leq \epsilon$.
- h is an ϵ-AXU family if $\forall x \neq y \in \mathcal{I}^l$, and $\forall z \in \mathcal{I}$: $\mathsf{Pr}_{\kappa \leftarrow K}[h_\kappa(x) \oplus h_\kappa(y) = z] \leq \epsilon$.
- If a UHF is linear, for any message pair (m_1, m_2): $h_\kappa(m_1) + h_\kappa(m_2) = h_\kappa(m_1 + m_2)$.

Error-Correcting Code (ECC)**.** An ECC [20] is used to express a sequence of the original data and the parity data such that any errors can be detected and corrected. An ECC has two parameters (n, l) where l denotes the number of the original blocks, and n denotes the number of blocks after adding $(n - l)$ redundant blocks. There exists (n, l)-ECC codes that can correct up to $t = \frac{n-l+1}{2}$ errors. The Reed-Solomon code (RS) [25] is a kind of ECC which uses a special polynomial: $g(x) = (x - a^i)(x - a^{i+1}) \cdots (x - a^{i+2t})$. The codeword of the RS code is $c(x) = g(x) \cdot i(x)$ where $g(x)$ is the generator polynomial over $\mathbb{F}_p$, $i(x)$ is the information block, and a is a primitive element of the field.
Encoder: The $2t$ parity symbols are given by: $p(x) = i(x) \cdot x^{n-l} \pmod{g(x)}$.
Decoder: Given a codeword $r(x)$ which is the original codeword $c(x)$ plus errors: $r(x) = c(x) + e(x)$, the RS decoder identifies the position and magnitude of up to t and corrects the errors.

- Calculating the syndrome: The RS codeword has $2t$ syndromes that depend on errors. The syndromes are calculated by substituting the $2t$ roots of $g(x)$ into $r(x)$.
- Finding the symbol error locations: This involves solving the equations with t unknowns. The first step is to find an error locator polynomial using the Berlekamp-Massey algorithm or the Euclid algorithm. The second step is to find the roots of this polynomial using the Chien search algorithm.
- Finding symbol error values: This involves solving the equations with t unknowns using the Forney algorithm.

Universal Hash Function which is constructed using the Reed-Solomon code (RS-UHF)**.** A RS-UHF [26] is constructed in a way as follows. Suppose that a message m is a vector $\overrightarrow{m} = (m_1, \cdots, m_l)$ where $m_i \in \mathcal{I}$ and suppose that an (n, l)-RS code over $\mathcal{I}$ is used. $\overrightarrow{m}$ is viewed as a polynomial representation of the form $p_{\overrightarrow{m}} = m_l x^{l-1} + m_{l-1} x^{l-2} + \cdots + m_1$. A RS code can be defined as a vector $\overrightarrow{k} = (k_1, \cdots, k_n)$. The codeword of a message $\overrightarrow{m}$ is the evaluation of polynomial $p_{\overrightarrow{m}}$ at point $(k_1, \cdots, k_n)$: $(p_{\overrightarrow{m}}(k_1), \cdots, p_{\overrightarrow{m}}(k_n))$. A UHF is $h_\kappa(m) = p_{\overrightarrow{m}}(\kappa)$ where κ is the key.

Message Authentication Code (MAC)**.** A MAC [27] is used to authenticate a message and to detect message tampering and forgery. A MAC is a tuple of (MGen, MTag, MVer):

- $\mathsf{MGen}(1^\lambda)$: generates a secret key κ given a security parameter λ.
- $\mathsf{MTag}_\kappa(m)$: computes a tag τ for the message m with the key κ.
- $\mathsf{MVer}_\kappa(m, \tau)$: outputs 1 if τ is a valid tag, and 0 otherwise.

Pseudo-random Function (PRF)**.** A PRF [28] is used to generate exponentially many random bits in a way that behaves like a random function. A PRF is a keyed family of a function $g : \mathcal{K}_{\mathsf{PRF}} \times \mathcal{L} \rightarrow \mathcal{I}$ which is indistinguishable from a random family of functions from $\mathcal{L}$ to $\mathcal{I}$. A PRF can be constructed from any pseudorandom generator as follows. Let $G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ be a length-doubling pseudorandom generator. Define $G_0 : \{0,1\}^n \rightarrow \{0,1\}^n$ such that $G_0(x)$ is the first n bits of $G(x)$. Define $G_1 : \{0,1\}^n \rightarrow \{0,1\}^n$ such that $G_1(x)$ is the last n bits of $G(x)$. For the key $\mathcal{K}_{\mathsf{PRF}} \in \{0,1\}^n$ and an input $x \in \{0,1\}^n$, the PRF is constructed as: $F_{\mathcal{K}_{\mathsf{PRF}}}(x) = G_{x_n}(G_{x_{n-1}}(\cdots G_{x_2}(G_{x_1}(\mathcal{K}_{\mathsf{PRF}}))\cdots))$ where $x_i \in \{0,1\}$ $(i = 1, \cdots, n)$ are the elements of x

$(x = \{x_1, \cdots, x_n\} \in \{0,1\}^n)$.

MAC based on Universal Hash Function (UMAC). A UMAC [26] can be constructed as the composition of a UHF with a PRF. Given a UHF family $h : \mathcal{K}_{\mathsf{UHF}} \times \mathcal{I}^l \to \mathcal{I}$ and a PRF family $g : \mathcal{K}_{\mathsf{PRF}} \times \mathcal{L} \to \mathcal{I}$, the UMAC is a tuple of UMAC = (UGen, UTag, UVer):

- $\mathsf{UGen}(1^\lambda)$: generates key (κ, κ') uniformly at random from $\mathcal{K}_{\mathsf{UHF}} \times \mathcal{K}_{\mathsf{PRF}}$.
- $\mathsf{UTag}_{\kappa,\kappa'}(m)$: works in space $\mathcal{K}_{\mathsf{UHF}} \times \mathcal{K}_{\mathsf{PRF}} \times \mathcal{I}^l \to \mathcal{L} \times \mathcal{I}$, outputs $(r, h_\kappa(m) + g_{\kappa'}(r))$ in which a unique counter $r \in \mathcal{L}$ is increased in each execution.
- $\mathsf{UVer}_{\kappa,\kappa'}(m, (c_1, c_2))$: works in space $\mathcal{K}_{\mathsf{UHF}} \times \mathcal{K}_{\mathsf{PRF}} \times \mathcal{I}^l \times \mathcal{L} \times \mathcal{I}$, outputs 1 if and only if $h_\kappa(m) + g_{\kappa'}(c_1) = c_2$.

Dispersal coding. The dispersal coding [14] is constructed as follows: To tag a message, the message is encoded under an (n, l)-RS code; and a PRF is then applied to the last s code symbols ($s \in \{1, \cdots, n\}$). A MAC is obtained on each of those s code symbols using UMAC. A codeword is valid if at least one of the last s symbols is the valid MAC.

- $\mathsf{KGenECC}(1^\lambda)$: selects key $\overrightarrow{\kappa} = \{\{\kappa_i\}_{i=1}^n, \{\kappa'_i\}_{i=n-s+1}^n\}$ randomly from space $\mathcal{K} = \mathcal{I}^n \times (\mathcal{K}_{\mathsf{PRF}})^s$. The keys $\{\kappa_i\}_{i=1}^n$ are used for the RS code. The keys $\{\kappa'_i\}_{i=n-s+1}^n$ are used for the PRF in the UMAC.
- $\mathsf{MTagECC}_{\overrightarrow{\kappa}}(m_1, \cdots, m_l)$: outputs $(c_1, \cdots, c_n)$ in which $c_i = \mathsf{RS\text{-}UHF}_{\kappa_i}(\overrightarrow{m})$ when $i \in \{1, \cdots, n-s\}$ and $c_i = \mathsf{UTag}_{\kappa_i,\kappa'_i}(m_1, \cdots, m_l) = (r_i, \mathsf{RS\text{-}UHF}_{\kappa_i}(\overrightarrow{m}) + g_{\kappa'_i}(r_i))$ when $i \in \{n-s+1, \cdots, n\}$.
- $\mathsf{MVerECC}_{\overrightarrow{\kappa}}(c_1, \cdots, c_n)$: strips off the PRF from $(c_{n-s+1}, \cdots, c_n)$ as: $c'_i = c_i - g_{\kappa'_i}(r_i)$ where $i \in \{n-s+1, \cdots, n\}$, and then decodes $(c_1, \cdots, c_{n-s}, c'_{n-s+1}, \cdots, c'_n)$ using the RS decoder to obtain the message $\overrightarrow{m} = (m_1, \cdots, m_l)$. If the RS decoder fails at the point $\{\kappa_i\}_{i=1}^n$ (when the number of corruptions is more than $\frac{n-l+1}{2}$), MVerECC outputs $(\perp, 0)$. If one of the last s symbols of $(c_1, \cdots, c_n)$ is a valid MAC on $\overrightarrow{m}$ under UMAC, MVerECC outputs $(\overrightarrow{m}, 1)$, otherwise it outputs $(\overrightarrow{m}, 0)$.

Because the dispersal coding uses the RS code in MTagECC to tag the message and uses the RS decoder in MVerECC to verify, the dispersal coding can prevent the small corruption attack.

3. Adversarial Model

This paper considers an adversary $\mathcal{A}$ as follows. $\mathcal{A}$ may control the servers by corrupting the servers and robbing all the privileges of the servers. If $\mathcal{A}$ has not corrupted a server, $\mathcal{A}$ cannot do anything because that all the data and the keys between the client and the servers are assumed to be transmitted via a secure channel. After $\mathcal{A}$ corrupts a server, $\mathcal{A}$ can modify/replace/forge data stored on that server and pretend to be a healthy server by providing a fake valid MAC tag to the client, can prevent the client from recovering the original file, and can perform the below four attacks (small corruption attack, large corruption attack, replay attack and pollution attack). A restriction of $\mathcal{A}$ is that $\mathcal{A}$ can control at most $(n-h)$ out of the n servers within any time step (called *epoch*). More concretely, after corrupting a server, $\mathcal{A}$ can perform as follows:

(i) access to the encode and check phases to output a codeword c such that $\mathcal{A}$ can pass the verification without being detected with an advantage defined as: $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ND-POR}} = \Pr[\kappa \leftarrow \mathsf{KGenECC}_\kappa(1^\lambda); c \leftarrow \mathcal{A}^{\mathsf{MTagECC}_\kappa(\cdot),\mathsf{MVerECC}_\kappa(\cdot)}: \mathsf{MVerECC}_\kappa(c) = (m, 1) \wedge m$ is not queried to $\mathsf{MTagECC}_\kappa(\cdot)]$

(ii) prevent F to be recovered in the repair phase

(iii) perform the following four attacks.

Small corruption attack. $\mathcal{A}$ corrupts at most a t-fraction of the file F with a small data unit, where $t = \frac{n-l+1}{2}$, in order to hide the data loss incidents. This applies to the servers that want to preserve their reputation. To prevent the small corruption attack, the ECC is used to detect and correct errors [5], [6].

Large corruption attack. $\mathcal{A}$ corrupts more than a t-fraction of the file F with a large data unit, where t is the same parameter as in the small corruption attack, to discard a significant fraction of the data. This applies to the servers who want to sell the storage resource to multiple clients. To prevent the large data corruption, the spot check method is proposed [1], [4] in which the client randomly samples small portions of the data. Then, the server returns a computation over these portions of the data to the client. The results are checked by MACs. The spot check can only prevent the large corruption attack but cannot prevent the small corruption attack [1], [3].

Replay attack. $\mathcal{A}$ tries to prevent the client from repairing the corruption by re-using the old coded blocks instead of the current coded blocks and providing these old coded blocks to the client in the repair phase. For example, the client encodes the augmented blocks $\{b_1, b_2, b_3\}$ into six coded blocks: $c_{11} = b_1$ and $c_{12} = b_2 + b_3$ (stored on the server S_1), $c_{21} = b_3$ and $c_{22} = b_1 + b_2$ (stored on the server S_2), $c_{31} = b_1 + b_3$ and $c_{32} = b_2 + b_3$ (stored on the server S_3). In epoch 1, suppose that S_3 is corrupted. In epoch 2, the client repairs S_3 by two new coded blocks: $c'_{31} = b_1 + b_2 + 2b_3$ and $c'_{32} = 2b_1 + b_2$. In the end of epoch 2, suppose that S_1 is corrupted. In epoch 3, S_1 is repaired by two new coded blocks: $c'_{11} = 3b_1 + 3b_2$ and $c'_{12} = 3b_2 + 3b_3$. At this time, $\mathcal{A}$ re-uses the old coded blocks c_{31} and c_{32} of S_3 instead of c'_{31} and c'_{32}. Thus, if S_2 is corrupted in epoch 4, the linear combination between the coded blocks of S_1 and S_2 is unable to repair S_2.

Pollution attack. $\mathcal{A}$ uses a valid data to avoid detection in the check phase, but provides an invalid data in the repair phase. For example, the client encodes the augmented blocks $\{b_1, b_2, b_3\}$ into six coded blocks: $c_{11} = b_1$ and

$c_{12} = b_2 + b_3$ (stored on the server S_1), $c_{21} = b_3$ and $c_{22} = b_1 + b_2$ (stored on the server S_2), $c_{31} = b_1 + b_3$ and $c_{32} = b_2 + b_3$ (stored on the server S_3). In the check phase, suppose that the corrupted server S_3 is detected. In the repair phase, S_3 is repaired by two new coded blocks: $c'_{31} = b_1 + b_2 + 2b_3$ and $c'_{32} = 2b_1 + b_2$. At this time, $\mathcal{A}$ corrupts S_1 without detection because this time is the repair phase, not the check phase. To repair S_3, suppose that the client requests coded blocks from S_1 and S_2. S_1 then provides invalid coded blocks to the client.

One of the contributions is to prevent the small corruption attack. The other three attacks are still prevented in this paper by using the same solution as the RDC-NC scheme. These are discussed in the security analysis (Sect. 5).

4. The Proposed ND-POR Scheme

Throughout this paper, the notations described in Table 1 are used.

In the ND-POR scheme, n servers are employed. The first l servers $\{S_1, \cdots, S_l\}$, called **NC-servers**, store the coded blocks $\{c_{ij}\}_{i\in\{1,\cdots,l\}, j\in\{1,\cdots,\beta\}}$. The last $(n-l)$ servers $\{S_{l+1}, \cdots, S_n\}$, called **DC-servers**, store the dispersal coding parity blocks $\{d_{ij}\}_{i\in\{l+1,\cdots,n\}, j\in\{1,\cdots,\beta\}}$. The structure of the ND-POR scheme is depicted in Fig. 2. The ND-POR scheme is now described via each phase of the POR as follows.

Table 1 List of Notations

C	client
F	original file
m	number of file blocks
n	number of servers
l	number of NC-servers
$n-l$	number of DC-servers
β	number of blocks stored on a server.
k	file block index ($k \in \{1, \cdots, m\}$)
i	server index ($i \in \{1, \cdots, n\}$)
j	coded block index in a server ($j \in \{1, \cdots, \beta\}$)
v_k	file block ($k \in \{1, \cdots, m\}$)
b_k	augmented block of v_k ($k \in \{1, \cdots, m\}$)
S_i	server ($i \in \{1, \cdots, n\}$)
c_{ij}	coded block ($i \in \{1, \cdots, l\}$ and $j \in \{1, \cdots, \beta\}$)
d_{ij}	dispersal coding parity block ($i \in \{l+1, \cdots, n\}$, $j \in \{1, \cdots, \beta\}$)
$\mathbb{F}_p$	finite field of a prime order p
z	length of v_k over $\mathbb{F}_p$
$\mathbb{F}_p^z$	a vector of length z over $\mathbb{F}_p$
w	$w = z + m$ (length of a coded block over $\mathbb{F}_p$)
$\mathbb{F}_p^w$	a vector of length w over $\mathbb{F}_p$
f	Pseudo-Random Function (PRF) $f : \{0,1\}^* \times \{0,1\}^\kappa \to \mathbb{F}_p$ where κ is the key length of f and κ should be large enough (e.g., 160)
$\|\|$	concatenate operator
S_y	corrupted server
S'_y	new server which is used to replace S_y
h	number of healthy servers used for data repair
v	number of rows used for spot checks
$\mathcal{A}$	adversary
s	number of segments in a coded block of the RDC-NC scheme
λ	security parameter
t	boundary of small corruption attack or ECC threshold ($t = (n-l+1)/2$)

4.1 Keygen

C generates the secret key: $\mathcal{K} = \{\mathcal{K}_{\mathsf{rtag}}, \mathcal{K}'_{\mathsf{rtag}}, \{\mathcal{K}_i, \mathcal{K}'_i\}_{i\in\{l+1,\cdots,n\}}, \mathcal{K}_{\mathsf{enc}}\}$ which are randomly chosen in $\{0,1\}^\kappa$.

4.2 Encode

$F = v_1 \| \cdots \| v_m$. $v_k \in \mathbb{F}_p^z$ where $k \in \{1, \cdots, m\}$. C creates m augmented blocks $\{b_1, \cdots, b_m\}$ in which $b_k \in \mathbb{F}_p^w$ ($k \in \{1, \cdots, m\}$) has the form as Eq. 1 in Sect. 2.3, and where $w = z + m$. Given a set of m augmented blocks, C computes $l\beta$ coded blocks c_{ij} using the linear combinations and stores them on the NC servers $\{S_1, \cdots, S_l\}$. C then encodes c_{ij} using the dispersal coding into a dispersal coding parity block d_{ij} for each row and stores them on the DC-servers $\{S_{l+1}, \cdots, S_n\}$. Namely, the encode phase is described as follows.

1. *C computes coded blocks from m augmented blocks:*

   For $\forall i \in \{1, \cdots, l\}$, $\forall j \in \{1, \cdots, \beta\}$:

   - Generate m coefficients $\alpha_{ijk} \overset{rand}{\leftarrow} \mathbb{F}_p$ where $k \in \{1, \cdots, m\}$.
   - Compute coded blocks: $c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} b_k$.

   Therefore, a matrix $\{c_{ij}\}_{i\in\{1,\cdots,l\}, j\in\{1,\cdots,\beta\}}$ is constructed.

2. *C computes dispersal coding parity blocks in each row:*

   For $\forall i \in \{l+1, \cdots, n\}$, $\forall j \in \{1, \cdots, \beta\}$, C computes $d_{ij} = \mathsf{MTagECC}_{\mathcal{K}_i, \mathcal{K}'_i}(c_{i1}, \cdots, c_{il})$.

3. *C computes metadata for coded blocks:*

   For $\forall i \in \{1, \cdots, l\}$:

   - Generate w values $\{\xi_1, \cdots, \xi_w\}$: $\xi_u = f_{\mathcal{K}_{\mathsf{rtag}}}(i \| u)$ where $u \in \{1, \cdots, w\}$.
   - For $\forall j \in \{1, \cdots, \beta\}$, $c_{ij} \in \mathbb{F}_p^w$ is viewed as a column vector of w symbols: $c_{ij} = (c_{ij1}, \cdots, c_{ijw})$ with $c_{iju} \in \mathbb{F}_p$ where $u \in \{1, \cdots, w\}$. C computes

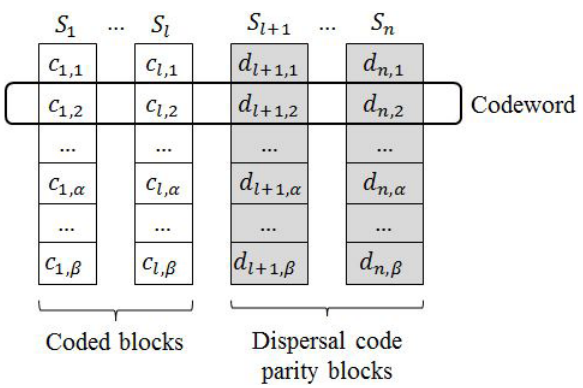

Fig. 2 The structure of the ND-POR scheme

a repair tag for c_{ij}: $T_{ij} = f_{\mathcal{K}'_{\mathsf{rtag}}}(i\|j\|\alpha_{ij1}\|\cdots\|\alpha_{ijm}) + \sum_{u=1}^{w} \xi_u c_{iju} \pmod p$.
- C encrypts the coefficients: $\epsilon_{ijk} = \mathsf{Enc}_{\mathcal{K}_{\mathsf{enc}}}(\alpha_{ijk})$ where $k \in \{1,\cdots,m\}$. This encryption is used to prevent the replay attack.

4. *C distributes data to the servers:*
   - C sends the coded blocks $\{c_{ij}\}_{i\in\{1,\cdots,l\},j\in\{1,\cdots,\beta\}}$, the encrypted coefficients $\{\epsilon_{ijk}\}_{i\in\{1,\cdots,l\},j\in\{1,\cdots,\beta\},k\in\{1,\cdots,m\}}$ and the repair tags $\{T_{ij}\}_{i\in\{1,\cdots,l\},j\in\{1,\cdots,\beta\}}$ to S_i where $i \in \{1,\cdots,l\}$.
   - C sends the dispersal coding parity blocks $\{d_{ij}\}_{i\in\{l+1,\cdots,n\},j\in\{1,\cdots,\beta\}}$ to S_i where $i \in \{l+1,\cdots,n\}$.

4.3 Check

C chooses a number of row indices to challenge the servers using the spot check method. The servers respond C. C checks the responses using the MVerECC algorithm. All the servers operate over the same subset of rows. Because the responses of all the servers lie on a codeword, all the servers can be checked for each challenge.

1. *C challenges the servers:* C firstly chooses an integer $v \overset{rand}{\leftarrow} [1,\beta]$. C then sends to each server a set of row indices $D = \{j_1,\cdots,j_v\}$ where $j_1,\cdots,j_v \overset{rand}{\leftarrow} [1,\beta]$ and a key $k \in \mathcal{I}$ where $\mathcal{I}$ is a field with operation $(+,\times)$.
2. *The servers respond C:* S_i computes: $R_i = \mathsf{RS\text{-}UHF}_k(c_{ij_1},\cdots,c_{ij_v})$.
3. *C verifies the servers:* Because all servers operate over the same subset of rows D, the combined response $R = (R_1,\cdots,R_n)$ is a codeword of the dispersal coding. C firstly checks R by calling $\mathsf{MVerECC}(R_1,\cdots,R_n)$ of the dispersal coding to verify. It returns false if the responses are invalid, and return true otherwise. After checking R, C checks the validity of each individual response R_i to detect which server is corrupted. For the l NC-servers $\{S_1,\cdots,S_l\}$, R_i is a valid response if it matches the i-th symbol in $\overrightarrow{m}$. For the $(n-l)$ DC-serves $\{S_{l+1},\cdots,S_n\}$, R_i is a valid response if it is a valid MAC on $\overrightarrow{m}$.

4.4 Repair

If a failure is detected in the check phase, C executes the repair phase with the following two sub-phases:

Sub-phase 1. The corruptions are firstly repaired by the RS decoder with the boundary number of corruptions $t = \frac{n-l+1}{2}$. If the number of corruptions is more than t, C uses the sub-phase 2.

Sub-phase 2. The corruptions are repaired by the network coding. C firstly requires the healthy servers to compute the aggregated coded blocks. Then, C combines these coded blocks to generate β coded blocks for the new server. Suppose that S_y is the corrupted server and S'_y is the new server which is used to replace S_y.

1. *C requests h healthy servers $\{S_{i_1},\cdots,S_{i_h}\}$ to compute the aggregated coded blocks and the proofs of correct encoding*:

   For $\forall i \in \{i_1,\cdots,i_h\}$:
   - C generates the coefficients $\{x_{i1},\cdots,x_{i\beta}\}$ where $x_{ij} \overset{rand}{\leftarrow} \mathbb{F}_p$ with $j \in \{1,\cdots,\beta\}$.
   - C requests S_i to compute an aggregated coded block and a proof of correct encoding.
   - S_i computes $\overline{a_i} = \sum_{j=1}^{\beta} x_{ij}c_{ij} \in \mathbb{F}_p^w$, then computes a proof of correct encoding: $\theta = \sum_{j=1}^{\beta} x_{ij}T_{ij} \pmod p$ and sends $\overline{a_i}, \theta, \{\epsilon_{ij1},\cdots,\epsilon_{ijm}\}_{j\in\{1,\cdots,\beta\}}$ to C.
   - C decrypts the encrypted coefficients from S_i to get the raw coefficients: $\{\alpha_{ij1},\cdots,\alpha_{ijm}\}_{j\in\{1,\cdots,\beta\}}$.
   - C re-generates w values $\{\xi_1,\cdots,\xi_w\}$: $\xi_u = f_{\mathcal{K}_{\mathsf{rtag}}}(i\|u)$ where $u \in \{1,\cdots,w\}$.
   - C checks if $\theta \neq \sum_{j=1}^{\beta} x_{ij} f_{\mathcal{K}'_{\mathsf{rtag}}}(i\|j\|\alpha_{ij1}\|\cdots\|\alpha_{ijm}) + \sum_{u=1}^{w} \xi_u a_{iu}$ where $\{a_{i1},\cdots,a_{iw}\}$ are the symbols of the block $\overline{a_i}$. This verification is to ensure that S_i does not have the pollution attack.
2. *C repairs S_y:*
   - C generates w values $\{\xi_1,\cdots,\xi_w\}$: $\xi_u = f_{\mathcal{K}_{\mathsf{rtag}}}(y\|u)$ where $u \in \{1,\cdots,w\}$.
   - For $\forall j \in \{1,\cdots,\beta\}$, $\forall \gamma \in \{1,\cdots,h\}$, C generates the coefficients $\alpha_{yj\gamma} \overset{rand}{\leftarrow} \mathbb{F}_p$, and computes the coded block: $c_{yj} = \sum_{\gamma=1}^{h} z_{yj\gamma}\overline{a_\gamma} \in \mathbb{F}_p^w$. By viewing c_{yj} as a column vector of w symbols: $c_{yj} = \{c_{yj1},\cdots,c_{yjw}\}$, C computes a repair tag for the block c_{yj}: $T_{yj} = f_{\mathcal{K}'_{\mathsf{rtag}}}(i\|j\|\alpha_{yj1}\|\cdots\|\alpha_{yjh}) + \sum_{u=1}^{w} \xi_u c_{yju} \pmod p$, and encrypts the coefficient: $\forall \gamma \in \{1,\cdots,h\}, \epsilon_{ij\gamma} = \mathsf{Enc}_{\mathcal{K}_{\mathsf{enc}}}(\alpha_{ij\gamma})$.
3. *C sends to the new server S'_y:*

   $\{c_{yj}\}_{j\in\{1,\cdots,\beta\}}, \{\epsilon_{yj\gamma}\}_{j\in\{1,\cdots,\beta\},\gamma\in\{1,\cdots,h\}}, \{T_{yj}\}_{j\in\{1,\cdots,\beta\}}$.

5. Security Analysis

This section describes the advantage of the defined adversary and explains how the small corruption attack, large corruption attack, replay attack and pollution attack are prevented.

5.1 Adversarial Check and Repair

A MAC consists of three algorithms: {MGen, MTag, MVer}.

Let q_1 denote the number of queries to MTag, and q_2 denote the number of queries to MVer. Let t denote the running time. The boundary of the advantage of $\mathcal{A}$ on UMAC [26] is given in the following fact:

Fact 1: Let $\mathsf{Adv}_{\mathsf{UMAC}}(q_1, q_2, t)$ denote the advantage of the adversary $\mathcal{A}$ on UMAC making q_1 queries to MTag, q_2 queries to MVer, and running in the time t. Let $\mathsf{Adv}_{\mathsf{prf}}(q_1, q_2, t)$ denote the advantage of the adversary $\mathcal{A}$ making $(q_1 + q_2)$ queries to the oracle PRF and running in the time t. Suppose that the UHF is an ϵ^{UHF}-AXU family of hash function. Then, the following inequality is obtained:

$$\mathsf{Adv}_{\mathsf{UMAC}}(q_1, q_2, t) \leq \mathsf{Adv}_{\mathsf{prf}}(q_1 + q_2, t) + \epsilon^{\mathsf{UHF}} q_2 \qquad (2)$$

Furthermore, the boundary of the advantage of $\mathcal{A}$ on the dispersal coding codeword is given as follows:

Theorem 1: Let $\mathsf{Adv}_{\mathsf{codeword}}(q_1, q_2, t)$ denote the advantage of $\mathcal{A}$ on the dispersal coding making q_1 queries to MTagECC, q_2 queries to MVerECC, and running in the time t. If RS-UHF is constructed from an (n, l)-RS code, then the following inequality is obtained:

$$\mathsf{Adv}_{\mathsf{codeword}}(q_1, q_2, t) \leq 2[\mathsf{Adv}_{\mathsf{UMAC}}(q_1, q_2, t)] \qquad (3)$$

Proof: Suppose that $\mathcal{A}$ is successful when $\mathcal{A}$ makes q_1 queries to the tagging oracle MTagECC, q_2 queries to the verification oracle MVerECC and runs in time t. $\mathcal{A}$ outputs a codeword $(c_1, \cdots, c_n)$ which can be decoded to the message $\overrightarrow{m} = (m_1, \cdots, m_l)$ such that at least one of the last s symbols in the codeword is a valid MAC on $\overrightarrow{m}$ computed with UMAC. Another adversary $\mathcal{A}'$ is considered for the UMAC construction. $\mathcal{A}'$ is given an access to a tagging oracle $\mathsf{UTag}_{\kappa,\kappa'}(\cdot)$ and a verification oracle $\mathsf{UVer}_{\kappa,\kappa'}(\cdot,\cdot)$ and needs to output a new message and a tag pair. $\mathcal{A}'$ chooses a position $j \in \{n - s + 1, \cdots, n\}$ randomly, and generates keys $\{\kappa_i\}_{i=1}^{n}$ and $\{\kappa'_i\}_{i=n-s+1}^{n}$ for $i \neq j$. $\mathcal{A}'$ runs $\mathcal{A}$. When $\mathcal{A}$ makes a query to tag $\overrightarrow{m} = (m_1, \cdots, m_l)$, $\mathcal{A}'$ computes $c_i \leftarrow \mathsf{RS\text{-}UHF}_{\kappa_i}(\overrightarrow{m})$ for $i \in \{1, \cdots, n - s\}$, and $c_i = \mathsf{UTag}_{\kappa_i,\kappa'_i}(\overrightarrow{m})$ for $i \in \{n - s + 1, \cdots, n\}, i \neq j$. $\mathcal{A}'$ calls the UTag oracle to compute $c_j = \mathsf{UTag}_{\kappa,\kappa'}(\overrightarrow{m})$. $\mathcal{A}'$ then responds to $\mathcal{A}$ with $\overrightarrow{c} \leftarrow (c_1, \cdots, c_n)$. When $\mathcal{A}$ makes a query $\overrightarrow{c} = (c_1, \cdots, c_n)$ to the verification oracle, $\mathcal{A}'$ tries to decode $(c_1, \cdots, c_{j-1}, c_{j+1}, \cdots, c_n)$ into message $\overrightarrow{m}$. If the decoding fails (the number of errors in the codeword is more than $t = \frac{n-l+1}{2}$), then $\mathcal{A}'$ responds to $\mathcal{A}$ with $(\perp, 0)$. Otherwise, let $\overrightarrow{m}$ be the decoded message. $\mathcal{A}'$ makes a query to the verification oracle $\alpha \leftarrow \mathsf{UVer}_{\kappa,\kappa'}(\overrightarrow{m}, c_j)$ and returns $(\overrightarrow{m}, \alpha)$ to $\mathcal{A}$. Assume that $\mathcal{A}$ outputs $\overrightarrow{c} = (c_1, \cdots, c_n)$ under the codeword that can be decoded to $\overrightarrow{m}$, such that $\overrightarrow{m}$ was not an input to the tagging oracle and at least one of the last s symbols in $\overrightarrow{c}$ is a valid MAC for $\overrightarrow{m}$. Then $\mathcal{A}'$ outputs $(\overrightarrow{m}, c_j)$. Because $t = \frac{n-l+1}{2}$, the number of remaining correct blocks is at least $n - \frac{n-l+1}{2} = \frac{n+l-1}{2}$. The number of correct parity blocks is thus at least $\frac{n+l-1}{2} - l = \frac{n-l-1}{2}$. Furthermore, the number of the original parity blocks before errors is $(n - l)$. Therefore, the number of correct parity blocks is at least $\frac{(n-l-1)/2}{n-l} \approx \frac{1}{2}$ of the number of the original parity blocks. In other words, the codeword $\overrightarrow{c}$ can be decoded if at least a majority of its parity blocks are correct. Then, with probability at least $\frac{1}{2}$, c_j is a correct MAC on $\overrightarrow{m}$. It follows that $\mathcal{A}'$ succeeds in outputting a correct message and MAC pair $(\overrightarrow{m}, c_j)$ with probability at least half the success probability of $\mathcal{A}$. □

From Eq. 2 and Eq. 3, the following inequality is obtained: $\mathsf{Adv}_{\mathsf{codeword}}(q_1, q_2, t) \leq \mathsf{Adv}_{\mathsf{prf}}(q_1 + q_2, t) + \epsilon^{\mathsf{UHF}} q_2$. Assume that the PRF is secure and the MAC is unforgeable, then $\mathsf{Adv}_{\mathsf{prf}}(q_1 + q_2, t) + \epsilon^{\mathsf{UHF}} q_2$ (negligible). Therefore, $\mathsf{Adv}_{\mathsf{codeword}}(q_1, q_2, t) \leq \varepsilon$. In other words, the probability for $\mathcal{A}$ to output codeword c such that $\mathcal{A}$ can pass the verification is negligible: $\mathsf{Pr}[\kappa \leftarrow \mathsf{KGenECC}_{\kappa}(1^{\lambda}); c \leftarrow \mathcal{A}^{\mathsf{MTagECC}_{\kappa}(\cdot),\mathsf{MVerECC}_{\kappa}(\cdot)} : \mathsf{MVerECC}_{\kappa}(c) = (m, 1) \wedge m \textit{ is not queried to } \mathsf{MTagECC}_{\kappa}(\cdot)] \leq \varepsilon$.

Now the probability of $\mathcal{A}$ to prevent data recovery is given as the following theorem.

Theorem 2: F can be recovered as long as in any epoch, at least h out of n servers are healthy and the matrix which consists of all the coefficients of the coded blocks has full rank, i.e., rank equals to m.

Proof: m augmented blocks are $\{b_1, \cdots, b_m\}$ which are created from m file blocks $\{v_1, \cdots, v_m\}$. The number of coded blocks is nx (n servers, x coded blocks per server). To compute a coded block c_{ij} for the server S_i, C chooses m coefficients $\{\alpha_{ij1}, \cdots, \alpha_{ijm}\}$, and uses the linearly independent combination: $c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} b_k$. $\{b_1, \cdots, b_m\}$ are viewed as the unknowns that need to be solved. After solving $\{b_1, \cdots, b_m\}$, the file blocks $v_1, \cdots, v_m$ can be obtained by picking the first coordinate of each b_k where $k \in \{1, \cdots, m\}$. F is finally recovered as $F = v_1 \| \cdots \| v_m$. To solve m unknowns $\{b_1, \cdots, b_m\}$, at least m coded blocks are required which make the matrix have full rank because the number of unknowns in an equation system has to be less than the number of equations. Let $\{c_{r_1}, \cdots, c_{r_m}\}$ denote such m coded blocks which are required for file recovery. Let $\{\alpha_{r_k 1}, \cdots, \alpha_{r_k m}\}$ denote m coefficients which are used to construct c_{r_k}.

$$\begin{cases} c_{r_1} = \sum_{k=1}^{m} \alpha_{r_1 k} b_k \\ c_{r_2} = \sum_{k=1}^{m} \alpha_{r_2 k} b_k \\ \cdots \\ c_{r_m} = \sum_{k=1}^{m} \alpha_{r_m k} b_k \end{cases}$$

Let h be the number of healthy servers that collectively store m coded blocks. In any epoch, $h = \frac{m}{x}$. In the RDC-NC scheme, there are n servers and α coded blocks per server. Thus, the number of healthy servers in an epoch in the RDC-NC scheme is at least $h = \frac{m}{\alpha}$. In the ND-POR scheme, there are also n servers but such n servers are divided into two types: l NC-servers and $(n - l)$ DC-servers. Because $l < n$, each NC-server has $\beta = \frac{n\alpha}{l}$ coded blocks. Therefore, the number of healthy servers in each epoch is at least $h = \frac{m}{\beta} = \frac{ml}{\alpha n}$. If the theorem is satisfied, the probability for $\mathcal{A}$ to prevent recovering F is negligible: $\mathsf{Pr}[F = \{v_1, \cdots, v_m\} \textit{ is not recovered}] \leq \varepsilon$. □

5.2 Small Corruption Attack

Theorem 3: The RS code in the dispersal coding is sufficient to prevent the small corruption attack.

Proof: Let t_{error} denote the number of corruptions caused by $\mathcal{A}$ in an epoch. Firstly, because the RS is constructed with the parameter (n, l), the message is interpreted as the description of a polynomial p of the degree less than l which is evaluated at n distinct points $\{a_1, \cdots, a_n\}$. The sequence of the values is the corresponding codeword C: $C = \{p(a_1), \cdots, p(a_n)\}$ (Sect. 2.4). Because any two different polynomials of the degree less than l agree in at most $(l-1)$ points, any two codewords of the RS code disagree in at least $n-(l-1) = n-l+1$ positions. Moreover, there are two polynomials that do agree in $(l-1)$ points but are not equal. Hence, the distance of the RS code is:

$$d = n - l + 1 \tag{4}$$

Secondly, because any two strings in C differ in at least d places, we have:

$$2t_{error} \leq d \tag{5}$$

From Eq. 4 and Eq. 5, $t_{error} \leq \frac{n-l+1}{2}$. The inequality reflects the fact that, given any string s, there is at most one string $c \in C$ which is within the distance d of t_{error} from s. This means that the advantage of $\mathcal{A}$ is always bounded by the error resilience of the RS code. □

5.3 Large Corruption Attack, Replay Attack and Pollution Attack

The attacks are addressed in the RDC-NC scheme. This section briefly describes the key ideas as follows:

Large corruption attack. In the check phase, using the spot check method, C periodically and randomly samples a set of indices of the coded blocks stored on each server. C then checks whether these sampled blocks match with the embedded MAC. If the adversary corrupts a large fraction of the data stored on the server, C easily detects the corruptions with an optimal computation and I/O at the server and communication between the server and the client.

Replay attack. To avoid the adversary $\mathcal{A}$ replaying a coded block, the common solution is to use a counter which is incremented each time the coded block on a server is recreated due to server failure. However, in this solution, the client must store locally the latest value of the counters. Therefore, the RDC-NC scheme uses a different solution to mitigate the replay attack and to reduce the storage cost for the client. That is, the coefficients are encrypted and stored together with the coded blocks to prevent $\mathcal{A}$ from knowing how the original blocks were combined to obtain the coded block. The ability of $\mathcal{A}$ is negligible because $\mathcal{A}$ does not know which old coded blocks to replay.

Pollution attack. For each coded block c_{ij} of the server S_i, a repair tag which is constructed from a MAC is embedded into the coded block. In the repair phase, C requires a number of servers which are used for data repair to provide their aggregated coded blocks. Before computing the new coded blocks for the new server, C uses the tag to check whether these servers combine the coded blocks correctly. Therefore, the servers cannot inject polluted blocks to C.

6. Efficiency Analysis

The Table 2 shows that the encode cost in the ND-POR scheme is more than that in the RDC-NC scheme. However, the encode phase is performed only one time in the beginning, but the check and repair phases are performed very often during the system lifetime. Therefore, the check and repair costs are more important than the encode cost. The Table 2 shows that the check and repair costs in the ND-POR scheme are less than these in the RDC-NC scheme.

Before analysing the costs in the RDC-NC and ND-POR schemes, recall that each coded block $c_{ij} \in \mathbb{F}_p^w$ where $w = m + z$. The coded block size is $w \log_2 p$. The unit of the below computational complexities is the number of operations over $\mathbb{F}_p^w$.

6.1 Encode Phase

The RDC-NC scheme computes $n\alpha$ coded blocks. Its encode computation cost is thus $O(n\alpha)$. The ND-POR scheme computes $l\beta$ coded blocks, then computes $(n-l)\beta$ dispersal coding parity blocks, where $\beta = \frac{n\alpha}{l}$. Its encode computation

Table 2 The comparison between the RDC-NC and ND-POR schemes

		RDC-NC	ND-POR
Security	Small corruption attack	No	Yes
	Large corruption attack	Yes	Yes
	Replay attack	Yes	Yes
	Pollution attack	Yes	Yes
Efficiency	Encode computation	$O(n\alpha)$	$O(n\alpha \times \frac{n}{l})$
	Check computation	$n\alpha$	$\frac{n\alpha}{l}$
	Repair computation	$O(\frac{3m\lvert F\rvert}{m+\alpha})$	$O(n \log_2^2 n \log_2 \log_2 n)$ (RS decode) $O(\frac{3lm\lvert F\rvert}{n\alpha+lm})$ (Network coding)
	The number of MACs	$ns\alpha$	$l\alpha$
	Required healthy servers	$\lceil \frac{m}{\alpha} \rceil$	$\lceil \frac{lm}{n\alpha} \rceil (l < n)$
	Storage server cost	$O(n\alpha \log_2 p(w+s+1))$	$O(n\alpha \log_2 p(\frac{nw}{l}+1))$

cost is thus $O(\frac{n^2\alpha}{l})$. It is clear that the cost in the ND-POR scheme is more than $\frac{n}{l}$ times that in the RDC-NC scheme. In an ECC, because the redundant blocks are chosen such that $n-l<l$, $\frac{n}{l}<2$. Because $n>l$, $\frac{n}{l}>1$. Therefore, $1<\frac{n}{l}<2$. This means that although the cost in the ND-POR scheme is more than $\frac{n}{l}$ times that of the RDC-NC scheme, it is less than double times.

The number of MACs. In the RDC-NC scheme, the number of MACs is $n\alpha s$ where n is the number of servers, α is the number of coded blocks stored on a server, s is the number of segments in a coded block. This is because the MACs are embedded in the segments of the coded blocks. In the ND-POR scheme, the number of MACs is $l\alpha$ where l is a number of servers out of n servers ($l<n$). This is because the MACs are only required for the network coding coded blocks which are located in only l servers.

6.2 Check Phase

The RDC-NC scheme challenges a subset of segment indices in each coded block of a server. $\mathcal{C}$ thus needs $n\alpha$ challenges to check all blocks stored on n servers where α is the number of blocks per server. In the ND-POR scheme, $\mathcal{C}$ challenges a subset of row indices (the codewords of the dispersal coding). Each codeword lies on all n servers. There are β codewords. $\mathcal{C}$ thus needs $\beta=\frac{n\alpha}{l}$ challenges to check all blocks stored on n servers. The cost in the RDC-NC scheme is more than l times that in the ND-POR scheme. This is an advantage of the ND-POR scheme when multiple servers are checked per challenge instead of one sever as the RDC-NC scheme.

6.3 Repair Phase

In the RDC-NC scheme, the cost for repairing a corrupted server is $O(h\frac{2|F|}{h+1}+\frac{|F|}{1+\frac{1}{h}})$ in which the cost of h healthy servers is $h\frac{2|F|}{h+1}$ and the cost of client-side is $\frac{|F|}{1+\frac{1}{h}}$). Because $h=\frac{m}{\alpha}$, the cost is $O(|F|\frac{3m}{\alpha+m})$. In the ND-POR scheme, in the sub-phase 1, the corruptions are repaired by the RS decoder which has $O(n\log_2^2 n\log_2\log_2 n)$ [29]. Because n is far less than the dominant parameter $|F|$, the cost of the RS decoder is less than the RDC-NC scheme. Let $n=12$ as in the experimental evaluation of the RDC-NC scheme, the RS can decode in only 284 field operations. In the sub-phase 2, the corruptions are repaired by the network coding like the RDC-NC scheme. The cost is thus $O(h'\frac{2|F|}{h'+1}+\frac{|F|}{1+\frac{1}{h'}})$. Because $h'=\frac{lm}{n\alpha}$, the cost in the ND-POR is thus $O(\frac{3lm|F|}{n\alpha+lm})$. Because $n>l$, $\frac{3m|F|}{m+\alpha}>\frac{3lm|F|}{n\alpha+lm}$. In both cases, the costs in the ND-POR scheme are still better than the RDC-NC scheme.

Parameter Choice. The parameter choice is now discussed for maximizing resilience of both cases simultaneously. Because an (n,l)-ECC can recover up to $t=\frac{n-l+1}{2}$ errors in each row, $\mathcal{A}$ can win the ECC if the number of corruptions is more than t. Furthermore, because the number of healthy servers is at least $\frac{ml}{n\alpha}$ (Theorem 2), $\mathcal{A}$ can win if $\mathcal{A}$ corrupts more than $\frac{ml}{n\alpha}$. Let $f_1(l)=\frac{n-l+1}{2}, f_2(l)=\frac{m}{\alpha}\times\frac{l}{n}$. l should be chosen such that the advantage of $\mathcal{A}$ is reduced. In other words, $f_1(l)$ and $f_2(l)$ are increased. If $f_1(l)$ and $f_2(l)$ are considered separately, $f_1(l)=\frac{n-l+1}{2}$ increases if l increases, $f_2(l)=\frac{ml}{n\alpha}$ increases if l decreases. It is not synchronous. Hence, l should be balanced between f_1 and f_2. Let $f_1(l)=f_2(l)$, we determine $l=\lceil\frac{n\alpha(n+1)}{2m+n\alpha}\rceil$.

Healthy Servers For Data Repair. As mentioned in Theorem 2, the RDC-NC scheme needs at least $\lceil\frac{m}{\alpha}\rceil$ healthy servers for data repair while the ND-POR scheme only needs at least $\lceil\frac{lm}{n\alpha}\rceil$ healthy servers for data repair ($l<n$). It is clear that the number of healthy servers in the RDC-NC scheme is more than $\frac{n}{l}$ times that in the ND-POR scheme.

6.4 Storage Cost

In the RDC-NC scheme, the size of $n\alpha$ coded blocks in $\mathbb{F}_p^w$ is $n\alpha w\log_2 p$. The size of $n\alpha s$ challenge tags in $\mathbb{F}_p$ is $n\alpha s\log_2 p$ where s denotes the number of segments in a coded block of the RDC-NC scheme. The size of $n\alpha$ repair tags in $\mathbb{F}_p$ is $n\alpha\log_2 p$. Therefore, the storage cost in the RDC-NC scheme is $O(n\alpha\log_2 p(w+s+1))$. In the ND-POR scheme, the size of $n\alpha$ coded blocks and $(n-l)\beta$ where $\beta=\frac{n\alpha}{l}$ dispersal coding parity blocks in $\mathbb{F}_p^w$ is $w\log_2 p(n\alpha+(n-l)\frac{n\alpha}{l})=\frac{n^2\alpha}{l}w\log_2 p$. The size of $n\alpha$ repair tags in $\mathbb{F}_p$ is $n\alpha\log_2 p$. Thus, the storage cost in the ND-POR scheme is $O(n\alpha\log_2 p(\frac{nw}{l}+1))$. To make the cost in the ND-POR scheme better than the RDC-NC scheme, let $n\alpha\log_2 p(\frac{nw}{l}+1)<n\alpha\log_2 p(w+s+1)$. As a result, the parameters should be chosen s.t. $w<\frac{sl}{n-l}$.

6.5 Numerical Examples of the Parameters

In this section, we give two concrete numerical examples of the parameters as follows.

Example 1. $n=12,\alpha=3,s=5,l=10,m=7,w=20,z=13,p=4099,|F|=1092$. Suppose that all elements in $\mathbb{F}_p$ is less than or equal to 4095. This is to let the elements not exceed 12 bits length. These parameters satisfy the conditions which we stated in the manuscript:

- $1<\frac{n}{l}<2$ as stated in Sect. 6.1.
- $l=\lceil\frac{n\alpha(n+1)}{2m+n\alpha}\rceil$ as stated in Sect. 6.3. (In this example, $l=\lceil\frac{12\cdot3\cdot13}{2\cdot7+12\cdot3}\rceil=10$).
- $w<\frac{sl}{n-l}$ as stated in Sect. 6.4.

We now show the costs of the RDC-NC and ND-POR schemes.

- The encode computation cost of the RDC-NC scheme is $n\alpha=12\cdot3=36$. Meanwhile, the encode computation cost of the ND-POR scheme is $n\alpha\frac{n}{l}=12\cdot3\cdot\frac{12}{10}=43.2$.

- The check computation cost of the RDC-NC scheme is $n\alpha = 12 \cdot 3 = 36$. Meanwhile, the check computation cost of the ND-POR scheme is $\frac{n\alpha}{l} = \frac{12 \cdot 3}{10} = 3.6$.
- The repair computation cost of the RDC-NC scheme is $\frac{3m|F|}{m+\alpha} = \frac{3 \cdot 7 \cdot 1092}{7+3} = 2293.2$. Meanwhile, the repair computation cost of the ND-POR scheme is $n \log_2^2 n \log_2 \log_2 n = 284$ (in the case of the RS code), or $\frac{3lm|F|}{n\alpha+lm} = \frac{3 \cdot 10 \cdot 7 \cdot 1092}{12 \cdot 3+10 \cdot 7} = 2163.4$ (in the case of the network coding).
- The number of MACs in the RDC-NC scheme is $ns\alpha = 12 \cdot 5 \cdot 3 = 180$. Meanwhile, the number of MACS in the ND-POR scheme is $l\alpha = 10 \cdot 3 = 30$.
- The number of the required healthy servers for data repair in the RDC-NC scheme is $\lceil \frac{m}{\alpha} \rceil = \lceil \frac{7}{3} \rceil = 3$. Meanwhile, that in the ND-POR scheme is $\lceil \frac{lm}{n\alpha} \rceil = \lceil \frac{10 \cdot 7}{12 \cdot 3} \rceil = 2$.
- The storage cost in the RDC-NC scheme is $n\alpha \log_2 p(w + s + 1) = 12 \cdot 3 \cdot \log_2 4099 \cdot (20 + 5 + 1) = 11232$. Meanwhile, the storage cost in the ND-POR scheme is $n\alpha \log_2 p(\frac{nw}{l} + 1) = 12 \cdot 3 \cdot \log_2 4099 \cdot (\frac{12 \cdot 20}{10} + 1) = 10800$.

Example 2. $n = 16, \alpha = 5, s = 4, l = 14, m = 10, w = 25, z = 15, p = 1031, |F| = 1500$. Suppose that all elements in $\mathbb{F}_p$ is less than or equal to 1023. This is to let the elements not exceed 10 bits length. These parameters satisfy the conditions which we stated in the manuscript:

- $1 < \frac{n}{l} < 2$ as stated in Sect. 6.1.
- $l = \lceil \frac{n\alpha(n+1)}{2m+n\alpha} \rceil$ as stated in Sect. 6.3. (In this example, $l = \lceil \frac{16 \cdot 5 \cdot 17}{2 \cdot 10+16 \cdot 5} \rceil = 14$).
- $w < \frac{sl}{n-l}$ as stated in Sect. 6.4.

We now show the costs of the RDC-NC and ND-POR schemes.

- The encode computation cost of the RDC-NC scheme is $n\alpha = 16 \cdot 5 = 80$. Meanwhile, the encode computation cost of the ND-POR scheme is $n\alpha \frac{n}{l} = 16 \cdot 5 \cdot \frac{16}{14} = 91.43$.
- The check computation cost of the RDC-NC scheme is $n\alpha = 16 \cdot 5 = 80$. Meanwhile, the check computation cost of the ND-POR scheme is $\frac{n\alpha}{l} = \frac{16 \cdot 5}{14} = 5.71$.
- The repair computation cost of the RDC-NC scheme is $\frac{3m|F|}{m+\alpha} = \frac{3 \cdot 10 \cdot 1500}{10+5} = 3000$. Meanwhile, the repair computation cost of the ND-POR scheme is $n \log_2^2 n \log_2 \log_2 n = 512$ (in the case of the RS code), or $\frac{3lm|F|}{n\alpha+lm} = \frac{3 \cdot 14 \cdot 10 \cdot 1500}{16 \cdot 5+14 \cdot 10} = 2863.64$ (in the case of the network coding).
- The number of MACs in the RDC-NC scheme is $ns\alpha = 16 \cdot 4 \cdot 5 = 320$. Meanwhile, the number of MACS in the ND-POR scheme is $l\alpha = 14 \cdot 5 = 70$.
- The number of the required healthy servers for data repair in the RDC-NC scheme is $\frac{m}{\alpha} = \frac{10}{5} = 2$. Meanwhile, that in the ND-POR scheme is $\frac{lm}{n\alpha} = \frac{14 \cdot 10}{16 \cdot 5} = 1.75$.
- The storage cost in the RDC-NC scheme is $n\alpha \log_2 p(w + s + 1) = 16 \cdot 5 \cdot \log_2 1031 \cdot (25 + 4 + 1) = 24000$. Meanwhile, the storage cost in the ND-POR scheme is $n\alpha \log_2 p(\frac{nw}{l} + 1) = 16 \cdot 5 \cdot \log_2 1031 \cdot (\frac{16 \cdot 25}{14} + 1) = 23657$.

In summary, although the ND-POR scheme combines the network coding and the dispersal coding, the ND-POR scheme is not worse than the RDC-NC scheme in totals.

7. Conclusion and Future Work

In this paper, the ND-POR scheme is proposed in which the network coding and the dispersal coding technique are combined to reduce the costs of two important phases, the check and the repair phases, and to prevent small corruption attack, replay attack, pollution attack and large corruption.

In future work, we focus on the following problem. The repair phase has not been optimized because the healthy servers need to provide the aggregated coded blocks to the client, then the client computes the new coded blocks and stores them on the new server. A new mechanism can be considered in which the healthy servers send their coded blocks directly to the new server without sending back to the client. This mechanism can reduce the burden for the client and also reduce the communication. To support this mechanism, a signature scheme can be employed such as [30], [31] that allows the new server to verify the coded blocks provided from the healthy servers instead of the client, and to construct the new coded blocks by itself.

References

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," Proc. 14th ACM Conf. on Comput. and commun. security - CCS 2007, pp.598–609, 2007.
[2] G. Ateniese, R. Di Pietro, L. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," Proc. 4th Conf. on Security and privacy in commun. networks - SecureComm 2008 Article no.9, 2008.
[3] A. Juels and B. Kaliski, "PORs: Proofs of retrievability for large files," Proc. 14th ACM Conf. on Computer and Communications Security - CCS 2007, pp.584–597, 2007.
[4] H. Shacham and B. Waters, "Compact Proofs of Retrievability," Proc. 14th Conf. on the Theory and Application of Cryptology and Information Security - ASIACRYPT 2008, pp.90–107, 2008.
[5] K. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: theory and implementation," Proc. Workshop on Cloud computing security - CCSW 2009, pp.43–54, 2009.
[6] 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 2009, pp.187–198, 2009.
[7] J. Hendricks, G.R. Ganger, and M. Reiter, "Verifying distributed erasure-coded data," Proc. 26th ACM symposium on Principles of

distributed computing - PODC 2007, pp.139–146, 2007.
[8] Z. Zhang, Q. Lian, S. Lin, W. Chen, Y. Chen, and C. Jin, "Bitvault: A highly reliable distributed data retention platform," Newsletter ACM SIGOPS Operating Systems Review, vol.41, no.2, pp.27–36, 2007.
[9] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession," Proc. 28th Conf. on Distributed Computing Systems - ICDCS 2008, pp.411–420, 2008.
[10] M.K. Aguilera, R. Janakiraman, and L. Xu, "Efficient fault-tolerant distributed storage using erasure codes," Tech. Rep., Washington University in St. Louis, 2004.
[11] R. Ahlswede, N. Cai, S.-Y.R. Li, and R. Yeung, "Network information flow," IEEE Trans. Information Theory, vol.46, no.4, pp.1204–1216, 2000.
[12] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," IEEE Trans. Information Theory, vol.56, no.9, pp.4539–4551, Sept. 2010.
[13] J. Li, S. Yang, X. Wang, X. Xue, and B. Li, "Tree-structured Data Regeneration in Distributed Storage Systems with Network Coding," Proc. 29th Conf. on Inform. commun., pp.2892–2900, 2010.
[14] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-based Distributed Storage Systems," Proc. ACM Cloud Comput. Security Workshop - CCSW 2010, pp.31–42, 2010.
[15] A. Le and A. Markopoulou, "NC-Audit: Auditing for network coding storage," Int. Symposium on Network Coding 2012, pp.155–160, 2012.
[16] 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. Computers, vol.63, no.1, pp.31–44, 2014.
[17] R. Curtmola, O. Khan, and R. Burns, "Robust remote data checking," Proc. 4th ACM int. workshop on Storage security and survivability - StorageSS 2008, pp.63–68, 2008.
[18] B. Chen, R. Curtmola, "Robust dynamic remote data checking for public clouds," Proc. of ACM conf. on Computer and communications security - CCS 2012, pp.1043–1045, 2012.
[19] B. Chen and R. Curtmola, "Auditable Version Control Systems," Network and Distributed System Security Symposium - NDSS 2014.
[20] J. Baylis, Error-Correcting Codes: A Mathematical Introduction, CRC Press, Boca Raton, FL, 1998.
[21] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-Based Integrity for Network Coding," Proc. 7th Conf. on Applied Crypt. and Network Security - ACNS 2009, vol.5536, pp.292–305, 2009.
[22] H. Handschuh and B. Preneel, "Key-Recovery Attacks on Universal Hash Function Based MAC Algorithms," Proc. 28th Conf. on Cryptology: Advances in Cryptology - CRYPTO 2008, pp.144–161, 2008.
[23] R. Koetter and M. Medard, "An algebraic approach to network coding," IEEE/ACM Trans. Networking, vol.11, no.5, pp.782–795, Oct. 2003.
[24] J.L. Carter and M.N. Wegman, "Universal Hash Functions," Journal of Computer and System Sciences, vol.18, no.2, pp.143–154, 1979.
[25] M. Riley and I. Richardson, An introduction to Reed Solomon codes: principles, architecture and implementation, Prentice-hall, 2001.
[26] V. Shoup, "On fast and provably secure message authentication based on universal hashing," Proc. 16th Cryptology Conf. on Advances in Cryptology - CRYPTO 1996, vol.1109, pp.313–328, 1996.
[27] D.R. Stinson, Cryptography - Theory and Practice, CRC Press, Boca Raton, 1995.
[28] O. Goldreich, S. Goldwasser, and S. Micali, "How to Construct Random Functions," Journal of the ACM, vol.33, no.4, pp.792–807, 1986.
[29] F. Didier, "Efficient erasure decoding of Reed-Solomon codes," arXiv:0901.1886v1 [cs.IT], 2009.
[30] D. Catalano, D. Fiore, and B. Warinschi, "Efficient network coding signature in the standard model," Proc. 15th Conf. on Practice and Theory in Public Key Cryptography - PKC 2012, vol.7293, pp.680–696, 2012.
[31] W. Yana, M. Yanga, L. Lia, and H. Fang, "Short signature scheme for multi-source network coding," Journal Computer Commun., vol.35, no.3, pp.344–351, 2012.
[32] K. Omote and T. Thao, "A new efficient and secure POR scheme based on network coding," Proc. 28th Conf. on Advanced Information Networking and Applications - AINA 2014, pp.98–105, 2014.

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 a PhD candidate in information science from Japan Advanced Institute of Science and Technology (JAIST).