Vol.2015-DPS-162 No.55
Vol.2015-CSEC-68 No.55
2015/3/6

S-POR: An Extremely Simple Network Coding-based Proof of Retrievability

KAZUMASA OMOTE[1,a)] TRAN PHUONG THAO[1,b)]

Abstract: Cloud computing is a service by which clients can outsource their data to reduce the burdens of data storage and maintenance. However, cloud providers are untrustworthy, which therefore introduce several security challenges: data availability, data integrity, and data confidentiality. Security of data confidentiality consists of cryptographic approach and information-theoretic approach. For availability, integrity and information-theoretic confidentiality, network coding-based POR (Proof of Retrievability) schemes have been proposed to allow the client to check whether the data stored in the servers is available and intact. In this paper, we propose an extremely simple network coding-based POR, named S-POR (S-POR: Simple network coding-based Proof of Retrievability). The implementation shows that the S-POR incurs very low computation cost for both client-side and server-side. Due to its simplicity, the S-POR is the most practical network coding-based POR for a real system, to the best of our knowledge.

1. Introduction

Since the amount of data is increasing exponentially, data storage and data management are troublesome tasks for the clients. To reduce the burdens for the data owners, remote storage system called cloud is proposed. A cloud is considered as a service which the clients can publish, access, manage and share their data remotely and easily from anywhere via the Internet. However, the shortcoming of this method is that a cloud storage provider is never trusted. The cloud system thus introduces three security challenges: data availability, data integrity and data confidentiality. Data confidentiality consists of two security approaches: cryptographic approach and information-theoretic approach. For data availability, integrity and information-theoretic confidentiality, the Proof of Retrievability (POR) [1], [2], [3], which is a challenge-response protocol between a client and a server, was proposed to allow the client to check whether the data stored in the servers is available, intact and always retrievable.

Related work. Based on the POR, the following techniques are commonly used:

- *Replication.* This approach was proposed [4], [5] to allow the client to store a file replica in 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 in each server.
- *Erasure coding.* Because the replication incurs high storage cost, this approach was applied [6], [7] to provide space-optimal data redundancy. Erasure coding can reduce storage cost because each server stores file blocks instead of the file replica like 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.
- *Obvious RAM (ORAM).* Recently, the ORAM is applied to the POR [8], [9]. Basically, this technique is used for privacy-preserving the data store in the servers. By using the ORAM structure, the servers cannot obtain the data access patterns when the client performs the data check. For the data repair, the ORAM-based POR embeds the erasure coding to repair the data when an error occurs. The ORAM structure leads to high storage cost because of the hierarchical storage layout, and high computation cost because of the shuffling procedure every a number of read operations.
- *Network coding.* To address the drawback of erasure coding, network coding is applied [10], [11] to improve the efficiency in data repair. The client does not need to reconstruct the entire file before generating new coded blocks as erasure coding. Instead, the coded blocks which are retrieved from the healthy servers are used to generate new coded blocks. Compare with the ORAM, the structure of the network coding is much simpler with no hierarchical storage, no shuffling procedure and no the drawback of the erasure coding. Therefore, this paper focuses on the network coding technique.

Network coding-based POR. Dimakis et al. [10] were the first applying network coding to the distributed storage system. Chen et al. [11] adapted the scheme of Dimakis et al. to propose the

[1] Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa, Japan 923-1211
[a)] omote@jaist.ac.jp
[b)] tpthao@jaist.ac.jp

Remote Data Checking for Network Coding-based distributed storage system (RDC-NC) scheme which provides an elegant data repair by recoding encoded blocks in healthy servers during repair. Li et al. [12] proposed a tree-structure data regeneration for the network coding to optimize network bandwidth by using a maximum spanning tree. Cao et al. [13] applied the Luby transform (LT) code for reducing the computation cost because the LT code is a special network code which works in the finite field of order two and only uses exclusive-OR (XOR) operation. H. Chen et al. [14] proposed the NC-Cloud scheme to improve the cost-effectiveness of repair using the functional minimum storage regenerating (FMSR) code and lighten the encoding requirement of storage nodes during repair. Le et al. proposed the NC-Audit scheme [15] in which a third party auditor (TPA) is employed and is delegated the responsibility to check the servers instead of the client. Furthermore, the NC-Audit also focuses on preventing the data leakage from the TPA. Omote et al. proposed [16] (Multisource and Direct repair for Network Coding-based POR) in which multiple clients can participate in the system. Furthermore, the direct repair feature is supported to reduce the burden for the client. The public authentication is also provided.

Contribution. This paper proposes the core network coding-based POR, named S-POR (Simple network coding-based POR). The S-POR has the following advantages:

- The S-POR prevents two most common attacks of a network coding-based POR: response replay attack and pollution attack.
- The S-POR is condensed as the core framework. It is thus the most simple network coding-based POR.
- The performance evaluation of the S-POR shows that it is very applicable in a real system. The S-POR can be easily and efficiently used in future works.

Roadmap. The backgrounds of the POR, network coding and homomorphic MAC are introduced in Section 2. The adversarial model is presented in Section 3. The S-POR is proposed in Section 4. The security and efficiency analyses are given in Section 5 and 6. The performance evaluation is described in Section 7. The conclusion is drawn in Section 8.

2. Preliminaries

2.1 Proof of Retrievability (POR)

To check whether the availability and integrity of the data stored in the cloud servers are satisfied, researchers proposed the POR [1], [2], [3] which is a challenge-response protocol between a client (verifier) and a server (prover), to ensure that data stored in the servers is always available, intact and retrievable. 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' in 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.

2.2 Network Coding

The network coding is firstly proposed in the network scenario [17], [18], and is then applied in the distributed storage system scenario [11].

Fundamental Concept. In the network scenario, suppose that a source wants to send a file F to a receiver via the network. The source firstly divides F into m blocks $\{v_1, \cdots, v_m\}$. $v_k \in \mathbb{F}_q^z$ where $k \in \{1, \cdots, m\}$ and $\mathbb{F}_q^z$ denotes a z-dimensional finite field over a prime q. The source 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 $\{w_1, \cdots, w_m\}$ denote the augmented blocks. w_k has the following form:

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

The source sends $\{w_1, \cdots, w_m\}$ as packets to the network. Suppose that an intermediate node in the network receives θ packets $\{w_{i_1}, \cdots, w_{i_\theta}\}$. The intermediate node generates θ coefficients $\alpha_1, \cdots, \alpha_\theta \in \mathbb{F}_q$ 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 can retrieve the augmented blocks from any set of m combinations. If $y \in \mathbb{F}_q^{z+m}$ is a linear combination of $w_1, \cdots, w_m \in \mathbb{F}_q^{z+m}$, 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. From the original file $F = \{v_1, \cdots, v_m\}$ ($v_k \in \mathbb{F}_q^z$), the client firstly creates m augmented blocks $\{w_1, \cdots, w_m\} \in \mathbb{F}_q^{z+m}$. The client then chooses m coefficients $\{\alpha_1, \cdots, \alpha_m\} \in \mathbb{F}_q$ and linearly combines m augmented blocks to create the coded blocks as $c = \sum_{k=1}^{m} \alpha_k . w_k \in \mathbb{F}_q^{z+m}$. The client stores the coded blocks in 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. R. Koetter et al. [19] proved that if the prime q 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. 2.2, from the augmented blocks $\{w_1, w_2, w_3\}$, the client chooses the coefficients to compute six coded blocks. The client stores two

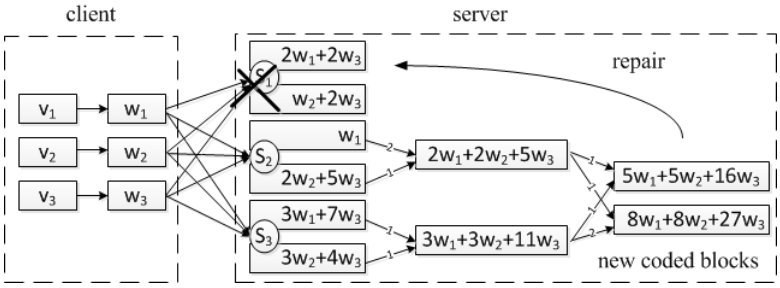

Fig. 1 An example of the network coding

coded blocks in 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.

2.3 Homomorphic MAC

The data stored in the servers cannot be checked alone. Instead, the data is embedded with an additional information, i.e., Message Authentication Code (MAC) or signature. A MAC is also call *tag*. The MAC approach is used in a symmetric key setting while the signature approach is used in an asymmetric key setting. Because this paper is based on a symmetric key setting for the efficiency, the MAC approach is used in this paper.

Traditional MAC. This MAC was proposed [20] to authenticate a message and to detect message tampering and forgery. This MAC consists of the following algorithms:

- $\mathsf{Gen}(1^\lambda) \rightarrow \{k\}$: on input a security parameter λ, the algorithm outputs a secret key k.
- $\mathsf{Tag}_k(M) \rightarrow t$: on input k and a message M, the algorithm computes a tag $t = M \cdot k$.
- $\mathsf{Ver}_k(M, t) \rightarrow \{0, 1\}$: on input M, t and k, the algorithm outputs 1 if t is a valid tag, and 0 otherwise.

Homomorphic MAC. The traditional MAC cannot be secured from the response replay attack (Section 3.1) when it is combined with the network coding. The homomorphic MAC was then introduced [17], [21] as a suitable technique for the network coding. The homomorphic MAC consists of the following algorithms:

- $\mathsf{Gen}(1^\lambda) \rightarrow \{k, k'\}$: on input a security parameter λ, the algorithm generates secret keys $\{k, k'\}$ which are used for tagging the message M, and for permuting the tag, respectively.
- $\mathsf{Tag}_{\{k,k'\}}(M) \rightarrow t$: on input $\{k, k'\}$ and a message M, the algorithm computes a tag $t = M \cdot k + f_{k'}(r)$ where f and r denote a pseudorandom function and a random, respectively.
- $\mathsf{Ver}_{\{k,k'\}}(M, t) \rightarrow \{0, 1\}$: outputs 1 if t is a valid tag, and 0 otherwise.

3. Adversarial Model

The S-POR scheme considers two common attacks of the POR: replay attack and pollution attack.

3.1 Response Replay Attack

This attack is performed by the servers. The malicious server re-uses the correct response in the past check phase in order to save the computation cost, and still passes the verification in the current check phase. An example is given as follows:

- Epoch 1: the client sends a challenge Q to the server $S_{\mathcal{A}}$. Based on Q, $S_{\mathcal{A}}$ then computes the response $\{c_{\mathcal{A}}, t_{\mathcal{A}}\}$ where $c_{\mathcal{A}}$ and $t_{\mathcal{A}}$ denote the aggregated coded block and aggregated tag, respectively. The client verifies the response by computing $t'_{\mathcal{A}} = c_{\mathcal{A}} \cdot k_C$ where k_C denotes the secret key of the client, and compares whether $t'_{\mathcal{A}} = t_{\mathcal{A}}$. Suppose that the equality occurs ($\{c_{\mathcal{A}}, t_{\mathcal{A}}\}$ is valid).
- Epoch 2: the client sends a challenge Q' to $S_{\mathcal{A}}$. $S_{\mathcal{A}}$ now re-uses $\{c_{\mathcal{A}}, t_{\mathcal{A}}\}$ in the epoch 1 without the need of computing another response for Q', and still passes the verification $t_{\mathcal{A}} = c_{\mathcal{A}} \cdot k_C$.

3.2 Pollution Attack

This attack is performed by the servers. The malicious server responses a valid coded block to pass the check phase, but then provides an invalid coded block in the repair phase. An example is given as follows:

- Encode: the client encodes the augmented blocks $\{w_1, w_2, w_3\}$ into six coded blocks: $c_{11} = w_1$ and $c_{12} = w_2 + w_3$ (stored in the server S_1), $c_{21} = w_3$ and $c_{22} = w_1 + w_2$ (stored in the server S_2), $c_{31} = w_1 + w_3$ and $c_{32} = w_2 + w_3$ (stored in the server S_3).
- Check: the corrupted S_3 is detected.
- Repair: to repair S_3, the client requires S_1 and S_2 to provide their aggregated coded blocks. S_1 provides $c_1 = 1c_{11} + 1c_12 = w_1 + w_2 + w_3$. Suppose S_2 injects a polluted block $c_2 = X$ which is an arbitrary value. The client computes two new coded blocks for the new server S'_3: $c'_{31} = 1c_1 + 1c_2 = X + w_1 + w_2 + w_3$, and $c'_{32} = 1c_1 + 2c_2 = 2X + w_1 + w_2 + w_3$.

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

4. The Proposed S-POR scheme

Throughout this paper, the followings notations are used.

C	client
F	original file
m	number of file blocks
n	number of servers
d	number of coded blocks per server
k	file block index, $k \in \{1, m\}$
i	server index, $i \in \{1, n\}$
j	coded block index per server, $j \in \{1, d\}$
v_k	file block ($k \in \{1, m\}$)
z	number of file blocks in $\mathbb{F}_q$
w_k	augmented vector of $\bar{w}_k$
S_i	server
c_{ij}	j-th coded block stored in S_i
t_{ij}	tag of c_{ij}
c_i	aggregated coded block of S_i
t_i	aggregated tag of S_i
s	number of spot check
l	number of healthy servers used for data repair
S_r	corrupted server
S'_r	new server replaced for S_r
$\mathbb{F}_q^z$	z-dimensional finite field $\mathbb{F}$ over a large integer q
f	Pseudo-random function: $\{0,1\}^* \times \{0,1\}^\kappa \to \mathbb{F}_q$
θ	number of pairs (c_i, t_i) that C can store.

The structure of the S-POR is given in Fig. 2. The S-POR is now described via each phase of the POR as follows.

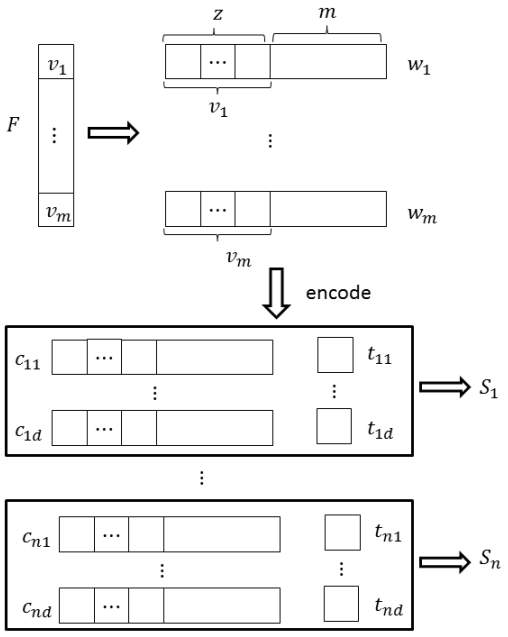

Fig. 2 The structure of S-POR

4.1 Keygen

C keeps two secret keys:

- $k_C \xleftarrow{rand} \mathbb{F}_q^{z+m}$.
- $k_{PRF} \xleftarrow{rand} \mathbb{F}_q$.

4.2 Encode

Step 1. C divides F and creates augmented blocks as follows:

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

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

.

Step 2. C computes $n \cdot d$ coded blocks and $n \cdot d$ tags as follows:
For $\forall i \in \{1, n\}, \forall j \in \{1, d\}$:

- C generates m coefficients: $\alpha_{ijk} \xleftarrow{rand} \mathbb{F}_q$.
- C computes coded block: $c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} \cdot w_k$.
- C computes tag: $t_{ij} = c_{ij} \cdot k_C + f_{k_{PRF}}(i||j)$.

Step 3. For $\forall j \in \{1, d\}$, C sends $\{c_{ij}, t_{ij}\}$ to S_i.

4.3 Check (Spot Check)

Step 1. C decides the values of s. This value can be different between the servers and can be changed every check time.

Step 2. C creates challenges as follows:

- Create s pairs $\{(b_1, \beta_1), \cdots, (b_s, \beta_s)\}$ where $b_u \xleftarrow{rand} [1, d]$ and $\beta_u \xleftarrow{rand} \mathbb{F}_q$ for $u \in \{1, s\}$.
- Send the challenge $Q = \{(b_1, \beta_1), \cdots, (b_s, \beta_s)\}$ to S_i.

Step 3. Each server combines its s coded blocks and s tags using the coefficient β_u and sends the aggregated coded block and the aggregated tag to C. Namely, S_i conducts the followings:

- Combine coded blocks: $c_i = \sum_{u=1}^{s} c_{ib_u} \cdot \beta_u$
- Combine tags: $t_i = \sum_{u=1}^{s} t_{ib_u} \cdot \beta_u$
- Send $\{c_i, t_i\}$ to C

Step 4. C verifies $\{c_i, t_i\}$ as follows:
$\forall i \in \{1, n\}$:

- C computes $t'_i = c_i \cdot k_C + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u)$.
- C checks whether $t_i = t'_i$.

Correctness of the verification:

$$\begin{aligned} t_i &= \sum_{u=1}^{s} t_{ib_u} \cdot \beta_u \\ &= \sum_{u=1}^{s} (c_{ib_u} \cdot k_C + f_{k_{PRF}}(i||b_u)) \cdot \beta_u \\ &= \sum_{u=1}^{s} c_{ib_u} \cdot k_C \cdot \beta_u + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u) \\ t'_i &= c_i \cdot k_C + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u) \\ &= \sum_{u=1}^{s} c_{ib_u} \cdot \beta_u \cdot k_C + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u) \\ &= t_i \end{aligned}$$

Note that in every epoch, C always keeps θ latest pairs of $\{c_i, t_i\}$, denoted as $R = \{(c_i, t_i)\}$ for $i \in \{i_1, i_\theta\}$. To reconstruct F, C retrieves $(m - \theta)$ pairs of aggregated coded blocks and tags from other healthy servers.

4.4 Repair

When a corrupted server (S_r) is detected, and the new server (S'_r) is used to replace S_r. There are two cases:

Case 1: $\theta > d$

Step 1. C picks randomly any d out of θ coded blocks in R. C computes their tags. Namely,

- Suppose these d coded blocks are: $\{c_{i_1}, \cdots, c_{i_d}\}$. The aggregated coded blocks now become the new coded blocks of the new server S'_r. This means that:

$$(c_{r_1}, \cdots, c_{r_d}) \leftarrow (c_{i_1}, \cdots, c_{i_d})$$

- For $j \in \{1, d\}$: C computes $t_{rj} = c_{rj} \cdot k_C + f_{k_{PRF}}(r||j)$.

Step 2. C sends $\{c_{rj}, t_{rj}\}$ to S'_r where $j \in \{1, d\}$.

Case 2: $\theta < d$

Step 1. C performs the check phase to collect $(d - \theta)$ more coded blocks.

Step 2. C gathers his θ coded blocks and $(d - \theta)$ coded blocks from Step 1. C then computes their tags. Namely,

- Suppose these d coded blocks are: $\{c_{i_1}, \cdots, c_{i_d}\}$. The aggregated coded blocks now become the new coded blocks of the new server S'_r. This means that:
$$(c_{r_1}, \cdots, c_{r_d}) \leftarrow (c_{i_1}, \cdots, c_{i_d})$$
- For $j \in \{1, d\}$: C computes $t_{rj} = c_{rj} \cdot k_C + f_{k_{PRF}}(r||j)$.

5. Security Analysis

5.1 Response Replay Attack

Suppose the malicious server which performs this attack is S_i.

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

The probability for S_i to pass the verification is the probability to guess k_{PRF}, which is $\frac{1}{q}$. If q is chosen large enough (i.e, 160 bits), this probability is negligible, which is $\frac{1}{2^{160}}$.

5.2 Pollution Attack

This attack does not happen in the case 1 because only C participates in the repair procedure. In the case 2, the malicious server S_i provides a polluted pair of $\{c_i, t_i\}$ in Step 1. The key point here is that C always checks every provided response even in the repair phase. S_i cannot pass the verification $t_i = c_i \cdot k_C + \sum_{u=1}^{s} \beta_u \cdot f_{k_{PRF}}(i||b_u)$ if c_i and t_i are not the independent linear combinations of the coded blocks and tags at the points: $\{b_1, \cdots, b_s\}$ using the coefficient $\{\beta_1, \cdots, \beta_s\}$.

6. Efficiency Analysis

The efficiency comparison between the RDC-NC [11], NC-Audit [15], MDNC [16] and S-POR schemes is given in Table 1. The NC-Audit and MDNC schemes support the public authentication in which the TPA is delegated the task of checking the servers by the client. For the fair comparison, suppose that the check phase in the NC-Audit and MDNC schemes is performed by the client. Furthermore, suppose that the system model in the MDNC only has a single client.

6.1 Storage Cost

Client. In the RDC-NC and MDNC, C has the keys in $\mathbb{F}_q^{z+m}$. The storage cost is thus $O((z + m) \log_2 q)$. In the NC-Audit, C stores the keys and mnd coefficients. The storage cost is thus $O((z + m + mnd) \log_2 q)$. In the S-POR, C stores the keys and θ coded blocks. The storage cost is thus $O((z+m) \log_2 q)+\theta(\frac{|F|}{m}+m)$.

Server. The size of a coded block equals that of an augmented block, which is $\frac{|F|}{m} + m$. In all the schemes, n servers store nd coded blocks. The storage cost is thus $O(nd(\frac{|F|}{m} + m))$.

6.2 Encode Phase

Computation (client). In the RDC-NC, MDNC and S-POR, C computes m tags, then linearly combines m augmented blocks and tags into nd coded blocks and new tags, respectively. The cost is thus $O(mnd)$. In the MDNC, C only computes m tags. The cost is thus $O(m)$.

Computation (server). In the RDC-NC, NC-Audit and S-POR, n servers do nothing, just receive coded blocks and tags from C. The cost is thus $O(1)$. In the MDNC, n servers linearly combine m received augmented blocks and tags to nd coded blocks and new tags. The cost is thus $O(mnd)$.

Communication. In the RDC-NC and S-POR, C transmits nd coded blocks and tags to n servers. The cost is thus $O(nd(\frac{|F|}{m}+m))$. In the NC-Audit, C transmits not only coded blocks, tags but also all coefficients. The cost is thus $O(nd(\frac{|F|}{m} + m) + mnd)$.

6.3 Check Phase

Computation (client). In all the schemes, C performs a verification for each of n servers. The cost is thus $O(n)$.

Computation (server). In all the schemes, each of n servers responds the aggregated coded block and tag of d stored coded blocks and tags, respectively. The cost is thus $O(nd)$.

Communication. In all the schemes, each of n servers transmit an aggregated coded block and tag to C. The cost is thus $O(n(\frac{|F|}{m} + m))$.

6.4 Repair Phase

Computation (client). In the RDC-NC, NC-Audit and MDNC, C computes d coded blocks and tags for the new server using the aggregated coded block and tag of each of l healthy servers. The cost is thus $O(ld)$. In the S-POR, C retrieves and verifies $(d - \theta)$ coded blocks, then computes d coded blocks and tags for the new server. The cost is thus $O(2d - \theta)$.

Computation (server). In the RDC-NC, NC-Audit and MDNC, l healthy servers combine d coded blocks and tags to provide to C. The cost is thus $O(ld)$. In the S-POR, $(d - \theta)$ servers combine d coded blocks and tags to provide to C. The cost is thus $O((d-\theta)d)$.

Communication. In the RDC-NC and MDNC, l healthy servers provide the aggregated coded blocks and tags to C. C then transmits d coded blocks and tags to the new server. The cost is thus $O((l + d)(\frac{|F|}{m} + m))$. In the NC-Audit, C also transmits all coefficients. The cost is thus $O((l+d)(\frac{|F|}{m}+m)+ld)$. In the S-POR, $(d-\theta)$ servers transmit the aggregated coded blocks and tags. Then, C transmits d coded blocks and tags to the new server. The cost is thus $O((2d - \theta)(\frac{|F|}{m} + m))$.

Table 1 Comparison

		RDC-NC [11]	NC-Audit [15]	MDNC [16]	S-POR (proposal)
Storage cost	Client	$O((z+m)\log_2 q)$	$O((z+m+mnd)\log_2 q)$	$O((z+m)\log_2 q)$	$O((z+m)\log_2 q)+\theta(\frac{\lvert F\rvert}{m}+m)$
	Server	$O(nd(\frac{\lvert F\rvert}{m}+m))$ (*)	$O(nd(\frac{\lvert F\rvert}{m}+m))$	$O(nd(\frac{\lvert F\rvert}{m}+m))$	$O(nd(\frac{\lvert F\rvert}{m}+m))$
Encode phase	Computation (client)	$O(mnd)$	$O(mnd)$	$O(m)$	$O(mnd)$
	Computation (server)	$O(1)$	$O(1)$	$O(mnd)$	$O(1)$
	Communication	$O(nd(\frac{\lvert F\rvert}{m}+m))$	$O(nd(\frac{\lvert F\rvert}{m}+m)+mnd)$	$O(mn(\frac{\lvert F\rvert}{m}+m))$	$O(nd(\frac{\lvert F\rvert}{m}+m)$
Check phase	Computation (client)	$O(n)$	$O(n)$	$O(n)$	$O(n)$
	Computation (server)	$O(nd)$	$O(nd)$	$O(nd)$	$O(nd)$
	Communication	$O(n(\frac{\lvert F\rvert}{m}+m))$	$O(n(\frac{\lvert F\rvert}{m}+m))$	$O(n(\frac{\lvert F\rvert}{m}+m))$	$O(n(\frac{\lvert F\rvert}{m}+m))$
Repair phase	Computation (client)	$O(ld)$	$O(ld)$	$O(ld)$	$O(2d-\theta)$
	Computation (server)	$O(ld)$	$O(ld)$	$O(ld)$	$O((d-\theta)d)$
	Communication	$O((l+d)(\frac{\lvert F\rvert}{m}+m))$	$O((l+d)(\frac{\lvert F\rvert}{m}+m)+ld)$	$O((l+d)(\frac{\lvert F\rvert}{m}+m))$	$O((2d-\theta)(\frac{\lvert F\rvert}{m}+m))$

7. Performance Evaluation

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

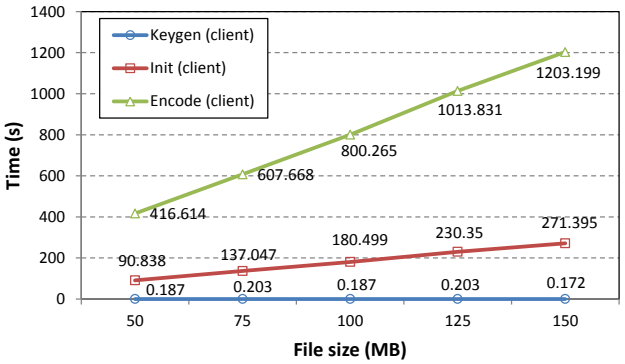

Fig. 3 The computation performance of the encode phase

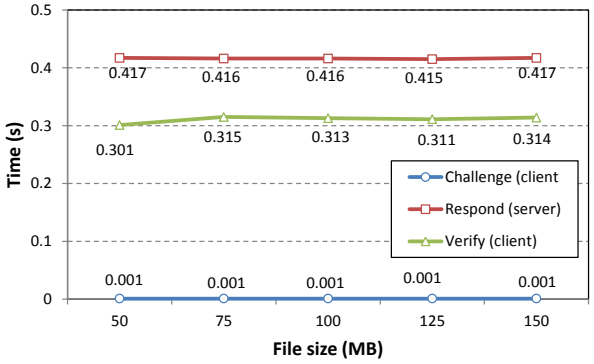

Fig. 4 The computation performance of the check phase

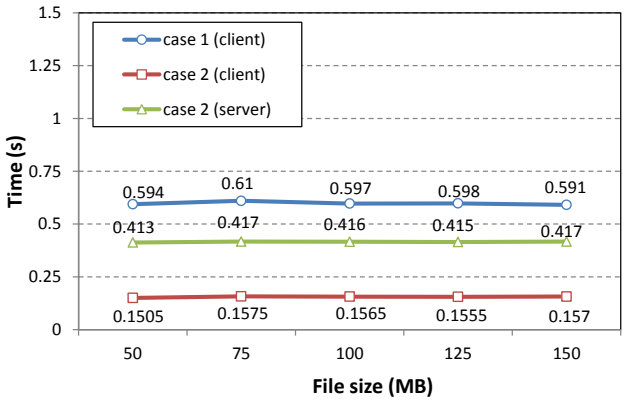

Fig. 5 The computation performance of the repair phase

In Fig. 3, the graphs reveal that the computation time in the encode and init (splitting file) functions linearly increases while the keygen function is constant. If the file size if 1 GB, the computation time in the encode and init functions is roughly 8308.561s and 1863.914s, respectively. In Fig. 5, the graphs reveal that the computation time of the check, response, and verify functions is constant. In Fig. 5, the graphs reveal that the computation time of the repair functions in both cases is also constant. The results show that the S-POR is very efficient and applicable in a real system.

8. Conclusion

This paper proposes the S-POR scheme, which is the core network coding-based POR. The security analysis proves that the S-POR can prevent the two most common attacks of a network coding-based POR: response replay attack and pollution attack. The evaluation results reveals that the S-POR is very fast and applicable in a real system. The S-POR can be easily used for further future works.

References

[1] 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.

[2] 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.

[3] K. Bowers, A. Juels and A. Oprea, "Proofs of retrievability: theory and implementation", *Proc. Workshop on Cloud computing security - CCSW 2009*, pp.43-54.

[4] W. J. Bolosky, J. R. Douceur, D. Ely and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of

desktop PCs", *Proc. of ACM conf. on Measurement and modeling of computation systems - SIGMETRICS*, 2000, pp.34-43.

[5] R. Curtmola, O. Khan, R. Burns and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession", *Proc. 28th Distributed Computation System Conf.*, 2008, pp. 411-420.

[6] M. K. Aguilera, R. Janakiraman and L. Xu, "Efficient fault-tolerant distributed storage using erasure codes", Tech. Rep., Washington University in St. Louis, 2004.

[7] K. Bowers, A. Juels and A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage", *Proc. 16th ACM Conf. on Computer and Communications Security - CCS 2009*, pp.187-198.

[8] E. Shi, E. Stefanov, and C. Papamanthou, "Practical dynamic proofs of retrievability", *Proc. of the 2013 ACM SIGSAC conf. on Computer and communications security - CCS 2013*, pp.325-336.

[9] D. Cash, A. Kp, and D. Wichs, "Dynamic Proofs of Retrievability via Oblivious RAM", *Advances in Cryptology EUROCRYPT 2013* Lecture Notes in Computer Science, vol.7881, 2013, pp.279-295.

[10] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright and K. Ramchandran, "Network coding for distributed storage systems", *IEEE Trans. on Information Theory*, 56(9):4539-4551, Sep 2010.

[11] 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.

[12] Jun Li, Shuang Yang, Xin Wang, X. Xue and Baochun Li, "Tree-structured Data Regeneration in Distributed Storage Systems with Network Coding", *Proc. 29th conference on Information communications - INFOCOM 2010*, pp. 2892-2900.

[13] Ning Cao, Shucheng Yu, Zhenyu Yang, Wenjing Lou and Y. Thomas Hou, "LT Codes-based Secure and Reliable Cloud Storage Service", *Proc. 31st IEEE conference on Computer Communications - INFOCOM 2012*.

[14] Henry C. H. Chen, Yuchong Hu, Patrick P. C. Lee and Yang Tang, "NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds", *Proc. of 10th USENIX conference on File and Storage Technologies - FAST 2012*, pp. 21-21.

[15] Anh Le and A. Markopoulou, "NC-Audit: Auditing for network coding storage", International Symposium on Network Coding - NetCod 2012, pp. 155-160.

[16] K. Omote, T. Thao, "MDNC: Multi-source and Direct Repair in Network Coding-based Proof of Retrievability Scheme", *Proc. of 15th Int. Workshop on Information Security Applications - WISA'14*, Springer-Verlag, 2014.

[17] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-Based Integrity for Network Coding", *Proc. 7th Conf. on Applied Crypt. and Network Security - ACNS 2009*, pp.292-305.

[18] R. Ahlswede, N. Cai, S. Li and R. Yeung, "Network information flow", *IEEE Trans. on Information Theory*, 46(4):1204-1216, 2000.

[19] R. Koetter and M. Medard, "An algebraic approach to network coding", *IEEE/ACM Trans. on Networking*, 11(5):782-795, Oct 2003.

[20] D. R. Stinson, Cryptography - Theory and Practice, CRC Press, Boca Raton, 1995.

[21] K. Izawa, A. Miyaji, and K. Omote, "Lightweight Integrity for XOR Network Coding in Wireless Sensor Networks", *Information Security Practice and Experience Lecture Notes in Computer Science*, vol.7232, 2012, pp.245-258.