実用的かつ効率的なネットワーク符号ベースのPOR方式

面 和成† トラン フン タオ ‡

† 北陸先端科学技術大学院大学
923-1292 石川県能美市旭台 1-1
{omote, tpthao}@jaist.ac.jp

あらまし PoR（Proofs of Retrievability の略）は，ストレージサーバ上のデータの完全性を検証することに加えて，完全性の検証に成功した箇所の部分データ（符号語）の取出しが保証される方式である．ネットワーク符号は，スループットや計算オーバヘッドを改善するために POR によく適用される．しかしながら，既存のネットワーク符号ベースの POR は，実用的かつ効率的な 2 つの特徴（直接修復と共通鍵暗号ベース）の両方を同時に達成できていなかった．本稿では，これらの問題を解決する新しい POR 方式を提案する．本方式は，あるサーバがダウンした時に，残りの健全なサーバがクライアントを利用せずに直接的に新しいサーバの分散符号データを作成できるため，クライアントはデータ修復の負荷から解放される．また本方式は，既存方式のような公開鍵暗号ベースではなく共通鍵暗号ベースであるため，非常に効率的である．

A practical and efficient network-coding-based POR scheme

Kazumasa Omote† Tran Phuong Thao†

†Japan Advanced Institute of Science and Technology (JAIST).
1-1 Asahidai, Nomi, Ishikawa 923-1211 JAPAN
{omote, tpthao}@jaist.ac.jp

Abstract POR (Proof of Retrievability) is a tool by which data owners can distribute their data to cloud servers, can check whether their data is intact and can repair a corrupted data. Using POR as a system model, network coding has been applied to improve network throughput and computation overhead in data check and data repair. Unfortunately, previous network-coding-based PORs have not achieved realistic and efficient features: direct repair and symmetric key setting. In this paper, we propose a new POR scheme addressing these gaps. When a server is detected as a corrupted server, our scheme supports direct repair in which the remaining servers send their distributed coded data directly to the new server instead of the client like prior works; the client thus is free from the burden of data repair. Furthermore, our scheme is based on symmetric key setting which is more efficient than asymmetric key setting as prior works.

1 Introduction

Data storage and management are the heavy tasks of data owners because data is increasing exponentially. Based on that demands, storage providers called clouds have been proposed to reduce the burdens of data owners. However, the problem is: cloud storages are untrusted in data integrity, availability and confidentiality. Ensuring data integrity and availability is a primary mission before ensuring confidentiality since they are the primary condition for the existence of a system.

Proof of Retrievability (POR). To enable clients to check data integrity and availability, researchers proposed Proof of Retrievability (POR) [4, 5] which is a challenge-response protocol between a client and a server. A POR consists of a tuple of (keygen, encode, check, repair).

Approaches. Using POR as a system model, many methods have been proposed. *Replication* [1] was proposed in which the client stores file replicas in each server. The client can periodically check and if a server is corrupted, the client will uses one replica to replace the corrupted one. The drawback of replication is the high storage cost for redundant replicas storage. To address this drawback, *erasure-coding* was proposed [10]. Instead of storing file replicas, the client stores file blocks in each server; the storage cost thus is reduced. However, the drawback of this approach is that the client must reconstruct the original file before repairing a corrupted server. The computation cost thus is increased during repair. To address that drawback, *network-coding* is proposed [2, 3] in which the client does not need to reconstruct the original file before repairing, instead he retrieves the data in the remaining healthy servers to generate the new data. Recently, there are several other approaches e.g., LT-code, Slepian-Wolf code, but they have not been used commonly. We thus focus on network coding.

MAC vs. signature. Data cannot be checked alone. Instead, the data is attached with an additional information: MAC tag (using symmetric key setting) and signature (using asymmetric key setting). Since the traditional MAC and signature are not suitable for network coding, new techniques called *homomorphic MAC* [11, 12] and *homomorphic signature* [14, 15] have been proposed. In this work, we choose symmetric key system because it is more efficient than asymmetric one. We thus focus on homomorphic MAC.

Network-coding-based POR. There are several notable schemes. Dimakis et al. [6] are the first to apply *network coding* to distributed storage systems to achieve a cost reduction in repair communication. Li et al. [7] proposed tree-structure linear network coding to achieve an efficient bandwidth capacity by using weighted maximum spanning tree. Chen et al. proposed RDC-NC [8] which provides an efficient data repair by recoding encoded blocks in the healthy servers. H.Chen et al. proposed NC-Cloud [9] to improve cost-effective repair using FMSR code. NC-Audit [16] was proposed for efficient checking and repair and data leakage using a combination of a homomorphic MAC called SpaceMac and a CPA-secure encryption called NCrypt.

Unfortunately, most of them have a same foible: When repairing a corrupted server, (i) the healthy servers provide their data to the client, (ii) the client checks and computes the new data and (iii) the client sends the new data to the new server. The client spends burdens during repair. We call this repair mechanism as *indirect repair*. Furthermore, the previous works still incur high storage, computation and communication costs although they mainly focus on efficiency. Therefore, our goal is to propose a scheme which can address these short-comings.

Contribution. We propose a new network-coding-based POR scheme which has the following advantages:

- Direct repair: during repair, the healthy servers provide their coded blocks and tags directly to the new server without sending them back to the client. The new server can check and computes the new coded blocks and tags for himself. The client thus is free from data repair.
- Symmetric key setting: herein introduces a difficulty: how to enable the new server (untrusted entity) to check and compute the new coded blocks/tags without using public key. In our scheme, we address this task by using the orthogonal keygen [13].
- Our storage, communication and computation costs are lower than these costs in prior schemes.

Organization We describe some preliminaries in Section II. We present our system and adversarial model in Section III and IV. We propose our scheme in Section V. We analyse security and efficiency in Section VI and Section VII. We conclude our work in Section VIII.

2 Preliminaries

2.1 Network coding (NC)

NC [2, 3] was proposed for cost-efficiency in data transmission and data repair. Suppose that a source S wants to send a file F to a receiver R. S divides F into m blocks: $F = \bar{w}_1||\cdots||\bar{w}_m$, each $\bar{w}_k \in \mathbb{F}_q^z$ where $k \in \{1, m\}$. S pads each $\bar{w}_k$ with a vector length m which contains a digit '1' in the position k and $m-1$ digit '0's elsewhere. Let w_k denote this augmented block.

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

S generates m coefficients and linearly combines m augmented blocks as: $c_S = \sum_{k=1}^{m} \alpha_k \cdot w_k$ where c_S is called a coded block. Suppose that S creates n coded blocks: $c_{S_1}, \cdots, c_{S_n}$. S then sends these to the network. When an intermediate node I receives t out of n packets, says $c_{k_1}, \cdots, c_{k_t}$. I generates t coefficients $\{\beta_1, \cdots, \beta_t\}$, linearly combines t received packets as: $c_I = \sum_{i=1}^{t} \beta_i \cdot c_{k_i}$ and transmit c_I to the adjacent node. Any coded block carries all accumulated coefficients which produce that coded block linearly independent to m file blocks. Therefore, when R combines the packets it receives (says c_R), the linear coefficients are contained in the last m elements of c_R. R thus can use these coefficients to recover m file blocks.

Application in POR [6, 7, 8, 9]. The system only has a client and multiple servers, instead of multiple nodes (source, intermediate and receiver node) as the network. The client divides his file into many blocks, makes augmented blocks, encodes

them and stores the coded blocks in the servers. For data check, each server provide only a *combined* coded block. For data repair, each healthy servers provide only a *combined* coded block. NC thus becomes an efficient method in data transmission and repair.

2.2 Orthogonal keygen

This technique [13] is proposed to generate a key which is orthogonal with all augmented blocks. In a formal statement, given m augmented blocks $w_1, \cdots, w_m$ and an integer p, the algorithm outputs a key k_p such that $k_p \cdot w_i = 0$ where $i = 1, \cdots, m$.
Building block. Let $w_1, \cdots, w_m \in \mathbb{F}_q^{z+m}$ be the augmented blocks that have span π. Let M be the matrix in which each of these m augmented blocks is a row of M. $\mathsf{rank}(M) = m$. Let π_M be the space spanned by the rows of M. The null space of M denoted as $\pi_M^{\perp}$ is the set of all vectors $u \in \mathbb{F}_q^{z+m}$ s.t. $M \cdot u^T = 0$. For any $m \times (z+m)$ matrix, the rank-nullity theorem gives: $\mathsf{rank}(M) + \mathsf{nullity}(M) = z + m$ where $\mathsf{nullity}(M)$ is the dimension of $\pi_M^{\perp}$. Thus, $\mathsf{dim}(\pi_M^{\perp}) = (z+m) - m = z$. Let $\{b_1, \cdots, b_z\} \in \mathbb{F}_q^{z(z+m)}$ be a basis of $\pi_M^{\perp}$ found by $M \cdot u^T = 0$. Let $\mathcal{P}$ be the space of p. Let PRF be a Pseudorandom function: $\mathcal{K} \times \mathcal{P} \times [1, z] \to \mathbb{F}_q$. k_p is computed as:

- $r \leftarrow PRF(k, p, i) \in \mathbb{F}_q, \forall i \in \{1, z\}$.
- $k_p \leftarrow \sum_{i=1}^{z} r \cdot b_i \in \mathbb{F}_q^{z+m}$.

3 System model

We use (POR) [4, 5] for our system model:

- $\mathsf{keygen}(s) \to \{\mathsf{sk}, \mathsf{pk}\}$: run by the client. Given a security parameter s, the algorithm outputs a pair of secret key sk and public key pk. For symmetric key setting, $\mathsf{pk} = \mathsf{null}$.
- $\mathsf{encode}(F_o) \to F_c$: run by the client. The algorithm divides the original file F_o into a number of file blocks, encodes them and stores the coded blocks (F_c) on the server.
- $\mathsf{check}() \to \{\mathsf{true/false}\}$: is an interact protocol between the client and the server. The client generates a challenge and sends it to the server. The server computes a response and sends it back to the client. The client finally verifies the server.
- $\mathsf{repair}() \to F_c'[S_{corr}]$: run by the client to repair a corrupted server S_{corr} with a new data $F_c'[S_{corr}]$. The specific technique of this algorithm depends on each concrete scheme.

The above is the original POR in which there is only one server and the repair phase is run by the client. To enable multiple servers and direct repair, we modify POR as follows:

- $\mathsf{keygen}(s) \to \{\mathsf{sk}, \mathsf{pk}\}$: similar to the original POR.
- $\mathsf{encode}(F_o) \to F_c$: similar to the original POR, but instead of storing the encoded data in a single server, the client store these in multiple servers.
- $\mathsf{check}() \to \{S_i, \mathsf{true/false}\}$: similar to the original POR but instead of checking one server, the client checks all servers.
- $\mathsf{repair}(S_r)$: run by the new server S_r'. Suppose S_r is a corrupted server, some remaining healthy servers will provide their coded blocks directly to S_r'. S_r'

checks these before computing the new coded block for itself.

4 Adversarial model

In our scheme, only the client is trusted. The other entities inside the system (i.e, all servers) and outside the system (e.g, a forger user/server) are untrusted. Concretely, they perform the following attacks:

Mobile attack run by untrusted entities (inside and outside the system). In each time step, the adversaries try to corrupt a number of servers as much as possible (let $P(\mathcal{A})$ be the ability of an adversary $\mathcal{A}$). Without being checked, he can destroy the system in $\frac{n}{P(\mathcal{A})}$ time steps where n denotes the number of servers.

Pollution attack run by the servers to prevent data recovering by injecting an invalid coded blocks in the repair phase. Concretely, in the check phase, the adversarial server $\mathcal{A}$ provides a correct response to the client. Suppose that another server S_B is detected as corrupted, the repair phase is executed. The remaining healthy servers are required to provide their coded blocks. Unfortunately if $\mathcal{A}$ is requested for repair because $\mathcal{A}$ has passed the check phase, $\mathcal{A}$ then provides an invalid coded block.

Curious attack run by the new server in the repair phase to learn the secret key of the client so that he can pass check phases in the later time steps.

5 Proposed scheme

To support direct repair in which the new server (untrusted) can check the provided coded blocks and tags without a public key, the key idea is that the key of the new server has to be orthogonal to all augmented blocks. Throughout our scheme, the notations in Table 5 are used.

表 1: Notations

$\mathcal{C}$	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, $j \in \{1, d\}$.
$\bar{w}$	file block.
w	augmented block.
t	MAC tag.
S_i	server.
c	coded block stored in the servers.
l	number of healthy servers.
$\mathbb{F}_q^z$	z-dimensional finite field $\mathbb{F}$ over a prime q.

$\mathcal{C}$ divides F into m file blocks $F = \bar{w}_1 || \cdots || \bar{w}_m$, each $\bar{w}_k \in \mathbb{F}_q^z$ where $k \in \{1, m\}$. $\mathcal{C}$ then creates m augmented blocks:

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

$\mathcal{C}$ uses his secret key to encoded these m augmented blocks into $n \cdot d$ coded blocks, and stores d coded blocks in each server. $\mathcal{C}$ checks each server periodically. If a corrupted server is detected, it will be repaired by some remaining healthy servers.

5.1 Keygen

Key for the client. Let k_C denote the secret key of $\mathcal{C}$. k_C is simply generated as: $k_C \overset{rand}{\leftarrow} \mathbb{F}_q^{z+m}$.

Key for each repair. Let p be the variant generated every repair time, $p \overset{rand}{\leftarrow} \mathbb{Z}_q^*, p \neq 1$. Using the orthogonal keygen, a key k_p is generated on input p and $\{w_1, \cdots, w_m\}$. The property of k_p is that: $k_p \cdot w_k = 0, \forall k \in \{1, m\}$. Let $k_r = k_C + k_p$. k_C is already computed in the beginning, only k_p is changed every repair time. k_r is sent to the new server only when data repair is executed.

5.2 Encode

Step 1: $\mathcal{C}$ computes a tag for each augmented block:
$\forall k \in \{1, m\}$, compute $t_{w_k} = w_k \cdot k_C$.

Step 2: $\mathcal{C}$ computes $n \cdot d$ coded blocks and $n \cdot d$ corresponding tags:
$\forall i \in \{1, n\}, \forall j \in \{1, d\}$:

- $\forall k \in \{1, m\}$, generate m coefficients: $\alpha_{ijk} \overset{rand}{\leftarrow} \mathbb{F}_q$.
- Compute code block: $c_{ij} = \sum_{k=1}^{m} \alpha_{ijk} \cdot w_k$.
- Compute tag: $t_{c_{ij}} = c_{ij} \cdot k_C$.

Step 3: $\mathcal{C}$ stores the coded blocks and the corresponding tags in the servers:
$\forall i \in \{1, n\}$, $\mathcal{C}$ sends d sets $\{c_{ij}, t_{ij}\}$ where $j \in \{1, d\}$ to S_i.

5.3 Check

$\mathcal{C}$ requires each server S_i to provide its combined coded block and tag for checking.

Step 1: The servers combine their coded blocks and tags:
$\forall i \in \{1, n\}$, S_i computes:

- $\forall j \in \{1, d\}$, generate d coefficients: $\beta_{ij} \overset{rand}{\leftarrow} \mathbb{F}_q$.
- Combine blocks: $c_{S_i} = \sum_{j=1}^{d} c_{ij} \cdot \beta_{ij}$.
- Combine tags: $t_{S_i} = \sum_{j=1}^{d} t_{c_{ij}} \cdot \beta_{ij}$.
- Send $\{c_{S_i}, t_{S_i}\}$ to $\mathcal{C}$.

Step 2: $\mathcal{C}$ verifies the servers:
$\forall i \in \{1, n\}$, $\mathcal{C}$ computes:

- Compute $t'_{S_i} = c_{S_i} \cdot k_C$.
- Check $t_{S_i} = t'_{S_i}$, return true ($\mathcal{S}_i$ is healthy) or false otherwise.

Correctness of the verification:
$t_{S_i} = \sum_{j=1}^{d} t_{c_{ij}} \cdot \beta_{ij} = \sum_{j=1}^{d} c_{ij} \cdot \beta_{ij} \cdot k_C$.
$t'_{S_i} = c_{S_i} \cdot k_C = \sum_{j=1}^{d} c_{ij} \cdot \beta_{ij} \cdot k_C$.

5.4 Repair

Suppose the corrupted server and the new server are denoted as S_r and S'_r.

Step 1: Any l servers provide their combined encoded blocks and tags to S'_r:
$\forall i \in \{1, l\}$, each S_i computes:

- $\forall j \in \{1, d\}$, generate d coefficients: $\beta_{ij} \overset{rand}{\leftarrow} \mathbb{F}_q$.
- Combine blocks: $c_{S_i} = \sum_{j=1}^{d} c_{ij} \cdot \beta_{ij}$.
- Combine tags: $t_{S_i} = \sum_{j=1}^{d} t_{ij} \cdot \beta_{ij}$.
- Send $\{c_{S_i}, t_{S_i}\}$ to S'_r.

Step 2: S'_r checks the provided data from S_i to prevent pollution attack from S_i:
$\forall i \in \{1, l\}$, S'_r computes:

- Compute $t'_{S_i} = c_{S_i} \cdot k_r$.
- Check whether $t'_{S_i} = t_{S_i}$.

The correctness of this verification is similar to that of the check phase, but k_r is used instead of k_C.

Step 3: S'_r computes d new coded blocks and tags:
$j \in \{1, d\}$, S'_r computes:

- $\forall i \in \{1, l\}$, generate l coefficients: $\gamma_{ri} \xleftarrow{rand} \mathbb{F}_q$.
- Combine blocks: $c_{rj} = \sum_{i=1}^{l} \gamma_{ri} \cdot c_{S_i}$.
- Combine tags: $t_{rj} = \sum_{i=1}^{l} \gamma_{ri} \cdot t_{S_i}$.

6 Security analysis

Mobile adversary First, to prevent $\mathcal{A}$ from corrupting all the server, there is a condition for $\mathcal{A}$: $P(\mathcal{A}) \leq l$ (*). This means: the number of healthy servers must be at least the ability of $\mathcal{A}$ in each time step. Second, to ensure $\mathcal{C}$ to be able to recover F, the parameters are chosen such that: $l \geq \frac{m}{d}$ (**) because to recover F, we view m file blocks $\bar{w}_1, \cdots, \bar{w}_m$ as variables that need to be solved; the number of healthy coded blocks is $l \cdot d$; the number of variables must be less than the number of given results: $m \leq ld$ and thus $l \geq \frac{m}{d}$. From (*) and (**), we have $P(\mathcal{A}) \leq \frac{m}{d}$.

Pollution attack In step 1 (repair phase), a malicious server $\mathcal{A}$ in a set of l servers sends an invalid packet $(c_{\mathcal{A}}, t_{\mathcal{A}})$ to S'_r. Since the servers are assumed to not collude with each other and $k_r \in \mathbb{F}_q^{z+m}$ is only known by S'_r. $\mathcal{A}$ can only pass the verification of S'_r in step 2 with a probability: $\mathsf{Pr}_{\mathcal{A}} = \frac{1}{q^{z+m}}$. If q is chosen large enough (e.g, 160 bits), the probability becomes negligible.

Curious adversary The new server is given $k_r = k_C + k_p \in \mathbb{F}_q^{z+m}$. The probability of the adversary $\mathcal{A}$ to learn k_C is $\frac{1}{q^{z+m}}$ (guess k_C directly or guess k_p then obtain k_C by $k_C = k_r - k_p$). If q is chosen large enough, the probability of $\mathcal{A}$ becomes negligible.

7 Efficiency analysis

There are some notable NC-based PORs in which RDC-NC [8] and NC-Audit [16] have the same scenario as our scheme at most. Another works address different tasks: [9] focuses on repair technique without another phases of POR and security analysis; [7] focuses on designing topology of the network; [6] was improved in [8]. Our comparison is given in Table 2. Due to the space constrain, we skip the explanation.

8 Conclusion

We propose a new POR scheme to support direct repair with symmetric key using the orthogonal key technique [13]. We show the better storage, communication and computation costs based on complexity theory.

参考文献

[1] R. Curtmola, O. Khan, R. Burns and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession", ICDCS'08.

[2] R. Ahlswede, N. Cai, S. Li and R. Yeung, "Network information flow", *IEEE Trans. Info. Theory*, vol.46, no.4, pp.1204-1216, 2000.

[3] S. Li, R. Yeung, and N. Cai, "Linear Network Coding", in *IEEE Trans. on Info. Theory*, vol.49, no.2, pp.371-381, 2003.

[4] A. Juels and B.Kaliski, "PORs: Proofs of retrievability for large files", CCS'07.

表 2: Overhead comparison

		RDC-NC [8]	**NC-Audit** [16]	**Our scheme**
Feature	Direct repair	No	Not completed	Yes
	Symmetric key	Yes	Yes	Yes
Storage	Server-side	$O(dn(\frac{\|F\|}{m}+m))$	$O(dn(\frac{\|F\|}{m}+m))$	$O(dn(\frac{\|F\|}{m}+m))$
	Client-side	$O(5(z+m)\log_2 q)$	$O((z+m)\log_2 q)$	$O((z+m)\log_2 q)$
Encode	Computation (server)	$O(1)$	$O(1)$	$O(1)$
	Computation (client)	$O(mdn)$	$O(mdn)$	$O(mdn)$
	Communication	$O(dn(\frac{\|F\|}{m}+m))$	$O(dn(\frac{\|F\|}{m}+m)+mdh)$	$O(dn(\frac{\|F\|}{m}+m))$
Check	Computation (server)	$O(dn)$	$O(dn)$	$O(dn)$
	Computation (client)	$O(n)$	$O(1)$	$O(dn)$
	Communication	$O(n(\frac{\|F\|}{m}+m))$	$O(n(\frac{\|F\|}{m}+m))$	$O(n(\frac{\|F\|}{m}+m))$
Repair	Computation (server)	$O(dl)$	$O(l+2dl)$	$O(l+2dl)$
	Computation (client)	$O(l+dl)$	$O(1)$	$O(1)$
	Communication	$O((l+d)(\frac{\|F\|}{m}+m))$	$O(l(\frac{\|F\|}{m}+m)+ld)$	$O(l(\frac{\|F\|}{m}+m))$

[5] H. Shacham and B. Waters, "Compact Proofs of Retrievability", ASIACRYPT'08.

[6] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright and K. Ramchandran, "Network coding for distributed storage systems", *IEEE Trans. Info. Theory*, vol.56, no.9, 2010, pp.4539-4551.

[7] J. Li, S. Yang, X. Wang, X. Xue and B. Li, "Tree-structured Data Regeneration in Distributed Storage Systems with Network Coding", INFOCOM'00.

[8] B. Chen, R. Curtmola, G. Ateniese and R. Burns, "Remote Data Checking for Network Coding-based Distributed Storage Systems", CCSW'10.

[9] H. Chen, Y. Hu, P. Lee, Y. Tang, "NCCloud: A Network-Coding-Based Storage System in a Cloud-of-Clouds", *IEEE Trans. on Computers*, vol.63, no.1, 2014, pp.31-44.

[10] Y. Dodis, S. Vadhan and D. Wichs, "Proofs of Retrievability via Hardness Amplification", TCC'09.

[11] S. Agrawal, D. Boneh, "Homomorphic MACs: MAC-Based Integrity for Network Coding", ACNS'09.

[12] C. Cheng, T. Jiang, "An Efficient Homomorphic MAC with Small Key Size for Authentication in Network Coding", in *IEEE Trans. on Computers*, vol.62, no.10, 2012, pp.2096-2100.

[13] A. Le, A. Markopoulou, "On detecting pollution attacks in inter-session network coding", INFOCOM'12.

[14] N. Attrapadung and B. Libert, "Homomorphic network coding signatures in the standard model", PKC'11.

[15] D. M. Freeman, "Improved security for linearly homomorphic signatures: a generic framework", PKC'12.

[16] A. Le and A. Markopoulou, "NC-Audit: Auditing for network coding storage", NetCod'12.