Title	SW-POR: A Novel POR Scheme Using Slepian-Wolf Coding for Cloud Storage
Author(s)	THAO, Tran Phuong; KHO, Lee Chin; LIM, Azman Osman
Citation	2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing, and IEEE 11th Intl Conf on and Autonomic and Trusted Computing, and IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UTC-ATC-ScalCom): 464-472
Issue Date	2014-12
Type	Conference Paper
Text version	author
URL	http://hdl.handle.net/10119/13471
Rights	This is the author's version of the work. Copyright (C) 2014 IEEE. 2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing, and IEEE 11th Intl Conf on and Autonomic and Trusted Computing, and IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UTC-ATC-ScalCom), 2014, 464-472. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Description	

SW-POR: A Novel POR Scheme using Slepian-Wolf Code for Cloud Storage

Tran Phuong Thao, Lee Chin Kho, Azman Osman Lim
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa, Japan 923-1292
Email: {tpthao, s1120203, aolim}@jaist.ac.jp

***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 integrity, data availability and data confidentiality. Since integrity and availability are prerequisite conditions for the existence of a system, researchers focus on these before confidentiality. To ensure integrity and availability, network coding-based POR (Proof of Retrievability) schemes have been proposed to enable the client to check whether the distributed data stored in the cloud servers is intact or not. However, most existing schemes have not achieved exact-repair feature and efficiency. In this paper, we propose a new POR scheme named SW-POR (Slepian-Wolf-based Proof of Retrievability) to address these gaps. Our scheme supports exact repair in which the repaired data is the same as the original corrupted data. Our scheme also can reduce the storage, computation, communication overheads and the size of a coded block. We proposed SW-POR based on Slepian-Wolf data compression code. We prove our security using entropy theory and implement SW-POR to show the costs.**

***Keywords*—proof of retrievability, network coding, Slepian-Wolf code**

I. Introduction

Many individuals and organizations outsource their data to cloud storage providers which allow clients to access, manage and share their data easily from anywhere via the Internet. However, cloud provider are untrustworthy, there are numerous security challenges: data availability, data integrity and data confidentiality. In this work, we focus on ensuring integrity and availability because they are pre-conditions for the existence of cloud system, they are thus more important than confidentiality.

Checking integrity and availability is mainly based on three techniques: *replication*, *erasure coding* and *network coding*. Replication, which allows the client to store file replicas in servers, was firstly proposed in [1]. When a corrupted server is detected, the client uses the healthy replicas to recover it. However, the drawback of this method is high storage cost because the client must store a whole file in each server. *Erasure coding* [3] was then applied to reduce storage cost. Erasure code allows the client to store file blocks in each server redundantly instead of file replica as *replication*. However, when recovering corrupted data, the client has to reconstruct the entire original file before generating new coded blocks. Therefore, its computation and communication costs are increased in data repair. To enable efficiency in repair phase, *network coding* has been applied [4]–[6] so that the client does not need to reconstruct the entire file before generating coded blocks, instead coded blocks are retrieved from healthy servers to generate new coded blocks.

To restore assurances eroded by cloud storage, researchers proposed a tool called POR (Proof Of Retrievability) as a system model [7]–[9]. Many schemes have been proposed based on the POR, e.g., [2], [20] using replication and [14]–[16] using *erasure code*. Due to the advantages of network coding compared to replication and erasure code as introduced before, in this work, we focus on several notable network coding-based POR schemes. Dimakis et al. [10] were the first to apply *network coding* to distributed storage systems and achieve a remarkable reduction in the communication overhead of the repair component. Li et al. [11] proposed tree-structure data regeneration with linear network coding to achieve an efficient regeneration traffic and bandwidth capacity by using undirected-weighted maximum spanning tree and Prim algorithms. Chen et al. then proposed RDC-NC [12] which provides an elegant solution for efficient data repair by recoding encoded blocks in the healthy servers. H. Chen et al. proposed NC-Cloud [13] to improve the cost-effectiveness of repair using the functional minimum-storage regenerating (FMSR) codes and relax the encoding requirement of storage nodes during repair.

Unfortunately, most of these have not addressed the following shortcomings: (i) the repaired coded block is not in exactly the same form as the corrupted coded block (Section II.B), (ii) the size of coded block are greater than or equal to the size of each file block and (iii) although most of prior works focus on efficiency, the models still incur high storage, computation and communication costs.

Contribution We propose a new POR scheme named SW-POR to address the above gaps. SW-POR has the following advantages:

- The size of a coded block in SW-POR is smaller than that in network coding.
- Exact-repair: the repaired coded block is exactly the same as the corrupted coded block unlike network coding. The additional overheads are unnecessary under exact-repair. Exact-repair also permits the code to be systematic.
- Our storage, communication and computation costs are lower than the costs in network coding.

Because in a real cloud system, the size of the original file is huge, SW-POR particularly becomes meaningful.

Organization We describe the preliminaries of POR, network coding, Slepian-Wolf code and the notations used in our scheme in Section II. We propose our scheme SW-POR in Section III. We analyse security and efficiency in Section IV and Section V. We present the result from the experiment in Section VI. We conclude our work in Section VII.

II. Preliminaries

A. Proof Of Retrievability (POR)

To check the server, researchers have proposed POR [7]–[9] as a system model which is a challenge-response protocol between client and server. A POR has a tuple of (keygen, encode, check, retrieve, repair) as follows:

- $\mathsf{keygen}(1^\lambda)$: is a probabilistic algorithm run by the client given a security parameter λ to produce secret key sk and public key pk (for symmetric key system, pk is set to be null).
- $\mathsf{encode}(\mathsf{sk}, F)$: The client encodes a raw file F to an encoded file F', then stores F' in the server.
- $\mathsf{check}(\mathsf{sk})$: is a challenge-response run between the client and server during which the client uses sk to generate a challenge c and sends c to the server. The server computes a corresponding response r and sends r back to the client. The client then verifies the server based on c and r.
- $\mathsf{retrieve}(k_0, \cdots, k_{m-1})$: The client runs this algorithm when he wants to retrieve his raw file F. The client requests m healthy servers $\mathcal{S}_{k_0}, \cdots, \mathcal{S}_{k_{m-1}}$ to provide their coded blocks. The client then computes the raw file F based on these m coded blocks.
- $\mathsf{repair}()$: If a corrupted server is detected in the check phase, this algorithm is executed by the client to repair the corrupted data. The technique of this phase depends on each concrete scheme.

In this work, we focus on encoding, retrieving and repairing. The check phase is beyond the scope of this paper. There are several existing schemes addressing checking. Two popular methods are to use homomorphic MAC (Message Authentication Code) [17]–[19] and homomorphic signature [27]–[29]. Because we do not deal with the check phase, we do not need any key for checking the servers; and thus, the keygen is not described in this work.

B. Network coding

Network coding [4]–[6] was proposed to improve network throughput and efficiency in data transmission and data repair. The source firstly partitions the message into m block $\bar{\mathbf{w}}_1, \cdots, \bar{\mathbf{w}}_m$. Each $\bar{\mathbf{w}}_i \in \mathbb{F}_q^l$ where $i \in \{1, m\}$ and $\mathbb{F}_q^l$ denotes a l-dimensional finite field over a prime q. The source then augments each $\bar{\mathbf{w}}_i$ with the vector of length m consisting of a digit $'1'$ in the i-th position and $'0'$ elsewhere. Let $w_1, \cdots, w_m$ be the augmented vectors. Each w_i has the form:

$$w_i = (\bar{\mathbf{w}}_i, \overbrace{\underbrace{0, \cdots, 0, 1}_{i}, 0, \cdots, 0}^{m}) \in \mathbb{F}_q^{l+m}$$

The source then sends $\{w_1, \cdots, w_m\}$ as packets to the network. When an intermediate node receives k packets $w_{i_1}, \cdots, w_{i_k}$, the node generates k coding coefficients $\alpha_1, \cdots, \alpha_k \in \mathbb{F}_q$ and linearly combines the received packets and transmits the resulting linear combination to the adjacent nodes. Therefore, each vector w carries m accumulated coding coefficients in $\mathbb{F}_q$ that produce w as a linear combination of all m original file blocks. Receivers receive combinations of the original vectors and can retrieve the original message from any set of m combinations. If $y \in \mathbb{F}_q^{l+m}$ is a linear combination of $w_1, \cdots, w_m \in \mathbb{F}_q^{l+m}$ then the linear combination coefficients are contained in the last m coordinates of y.

Application in distributed systems [10]–[13] From m augmented blocks $w_1, \cdots, w_m$, the client chooses m coding coefficients $\alpha_1, \cdots, \alpha_m$ and computes coded blocks as $c = \sum_{i=1}^{m} \alpha_i \cdot w_i$, then stores those coded blocks in the servers. When corruption is detected, the client retrieves coded blocks from h healthy servers and linearly combines them to regenerate new coded blocks. An example of this mechanism is given in Figure 1.

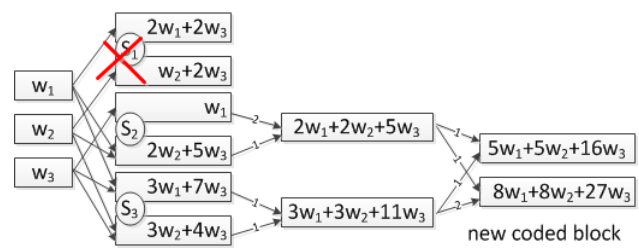


Fig. 1: *From augmented vectors $\{w_1, w_2, w_3\}$, the client computes six coded blocks and stores two coded blocks in the server $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$. Suppose $\mathcal{S}_1$ is corrupted, the client requires $\mathcal{S}_2$ and $\mathcal{S}_3$ to create new blocks using linear combination, and then mixes these to obtain two new coded blocks.*

In Figure 1, we can observe that the new coded blocks ($5w_1 + 5w_2 + 16w_3$ and $8w_1 + 8w_2 + 27w_3$) are not the same as the corrupted coded blocks ($2w_1 + 2w_3$ and $_2 + 2w_3$). Furthermore, since each file block is augmented with a vector length m, the size of each augmented block w_i is: $|w_i| = m + |\bar{\mathbf{w}}_i| = m + \frac{|F|}{m}$. The size of a coded block $\mathbf{c}_i$ is equal to the size of a augmented vector because the linear combination works in $\mathbb{F}_q$: $|\mathbf{c}_i| = |w_i|$. Therefore, $|\mathbf{c}_i| = m + \frac{|F|}{m}$. We will use this note in our efficiency analysis (Section V).

C. Slepian-Wolf coding (SWC)

We use the *binning idea* of SWC [24]. Assume that the source has two blocks X and Y which have the same bit-size. X is partitioned into a number of bins. During encoding, the index of the bin that the input lies into will be transmitted to the decoder instead of the input itself. Suppose we partition $|X|$ into t bins each with $\frac{|X|}{t}$ elements. Without SWC, we need $\log_2|X|$ bits to convey the input to the decoder. With SWC, we only need $\log_2 t$ bits instead. The decoder cannot tell what is the actual input with the bin index alone but Y now comes to the rescue. The decoder picks the element in the bin that is best matched with Y.

D. Notations

Throughout our scheme, the following notations are used:

$\mathcal{C}$	client.
F	original file.
m	number of file blocks.
$\bar{\mathbf{w}}_i$	file block ($i \in \{0, m-1\}$).
n	number of servers.
$\mathcal{S}_i$	server ($i \in \{0, n-1\}$).
$\mathbf{c}_i$	coded block stored in $\mathcal{S}_i$.
v_i	XOR constructing $\mathbf{c}_i$.
$\hat{\mathbf{c}}_i$	metadata of $\mathbf{c}_i$ (number of bit '1' of v_i).
α	first-operand index of v_i.
β	second-operand index of v_i.
γ	first-operand index of v_i.
$\lvert x \rvert$	bit-size of x
$\lvert \bar{\mathbf{w}} \rvert$	bit-size of each file block $\bar{\mathbf{w}}_i$, $\lvert \bar{\mathbf{w}} \rvert = \frac{\lvert F \rvert}{m}$
$\oplus$	exclusive-OR operator.
$H(x)$	Shannons entropy of a random variable x.

III. Proposed scheme

In SW-POR, the form of a coded block $\mathbf{c}_i$ is different from that of network coding. We use the key idea that each coded block is the *bin index* as in SWC to achieve better coded block size.

A. Encode

TABLE I: Encode algorithm (Encode)

```
INPUT: m, n, {w̄_0, ··· , w̄_{m-1}}, |F|
OUTPUT: {c_0, ··· , c_{n-1}}, {ĉ_0, ··· , ĉ_{n-1}}
01: count ← 0
02: for α ← 0 to m − 3
03:     for β ← α + 1 to m − 2
04:         for γ ← β + 1 to m − 1
05:             ĉ_i ← 0
06:             for x ← 0 to |w̄_α| − 1
07:                 if (w̄_α ⊕ w̄_β ⊕ w̄_γ == 1)
08:                     ĉ_i + +
09:             P_i ← list_all_combinations(|F|/m, ĉ_i)
10:             v_i ← w̄_α ⊕ w̄_β ⊕ w̄_γ
11:             for x ← 0 to P_i.length − 1
12:                 if (P_i[x] == v_i)
13:                     c_i ← x
14:                     break
15:             count + +
16:             if (count == n − 1)
17:                 break
18: return {c_0, ··· , c_{n-1}}, {ĉ_0, ··· , ĉ_{n-1}}
```

From m file blocks, there are $\binom{m}{3}$ XORs of any three different file blocks. However, we only need n out of $\binom{m}{3}$ XORs. The idea for choosing such n XORs is that:

(*) *$\mathcal{C}$ chooses $\bar{\mathbf{w}}_0$ as the first operand, $\bar{\mathbf{w}}_1$ as the second operand and $\bar{\mathbf{w}}_2, \cdots, \bar{\mathbf{w}}_{m-1}$ as the third operands, respectively. $\mathcal{C}$ then chooses $\bar{\mathbf{w}}_0$ as the first operand, $\bar{\mathbf{w}}_2$ as the second operand and $\bar{\mathbf{w}}_3, \cdots, \bar{\mathbf{w}}_{m-1}$ as the third operands, respectively. $\mathcal{C}$ repeats this process until there are enough n XORs. The first-operand index, the second-operand index and the third-operand index of the XOR used to construct a coded block are decided, unlike network coding in which the coded blocks are distributed randomly to the servers. The coded block in $\mathcal{S}_0, \mathcal{S}_1, \cdots, \mathcal{S}_{n-1}$ are constructed from $\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2, \cdots, \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_{m-1}, \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_3, \cdots, \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_{m-1}, \cdots, \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_3, \cdots, \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_{m-1}, \cdots$, respectively*. To reconstruct F when a server is corrupted, n, m are chosen as $2m < n \leq \binom{m}{3}$. n can be reduced if the scheme is extended so that each $\mathcal{S}_i$ stores multiple coded blocks.

The Encode algorithm is described in Table I. On input m, n, $\{\bar{\mathbf{w}}_0, \cdots, \bar{\mathbf{w}}_{m-1}\}$ and $|F|$, the algorithm outputs n coded blocks $\mathbf{c}_0, \cdots, \mathbf{c}_{n-1}$ and their metadata $\hat{\mathbf{c}}_0, \cdots, \hat{\mathbf{c}}_{n-1}$. i denotes the coded block index, α denotes the first-operand index, β denotes the second-operand index and γ denotes the third-operand index of a XOR. Our coded block are not simply the XORs. Firstly, $\mathcal{C}$ finds the number of bit '1' ($\hat{\mathbf{c}}_i$) when computing $\bar{\mathbf{w}}_\alpha \oplus \bar{\mathbf{w}}_\beta \oplus \bar{\mathbf{w}}_\gamma$ (line 5-8). Secondly, $\mathcal{C}$ constructs a vector P_i which consists of all possible values of each XOR (line 9) using *list_all_combination* (this common function is supported in many libraries in programming language) given the length of each file block $\frac{|F|}{m}$ and $\hat{\mathbf{c}}_i$. Thirdly, $\mathcal{C}$ finds the corresponding index of $\bar{\mathbf{w}}_\alpha \oplus \bar{\mathbf{w}}_\beta \oplus \bar{\mathbf{w}}_\gamma$ in P_i, returns to $\mathbf{c}_i$ (line 10-14). Our final coded blocks are the index of the XORs in their corresponding sets. The number of elements in P_i is $|P_i| = \binom{|F|/m}{\hat{\mathbf{c}}_i}$ and the length of each coded block is $\log_2 |P_i|$. The bandwidth and the storage cost can be reduced since the length of an index is less than the length of a coded block. Then, the Encode algorithm returns the list of coded blocks $\{\mathbf{c}_0, \cdots, \mathbf{c}_{n-1}\}$, the list of the metadata $\{\hat{\mathbf{c}}_0, \cdots, \hat{\mathbf{c}}_{n-1}\}$. Finally, $\mathcal{C}$ distributes $\{\mathbf{c}_i, \hat{\mathbf{c}}_i\}$ to $\mathcal{S}_i$ where $i = 0, \cdots, n-1$.

Example 1: Suppose all operations work in $\mathbb{F}_2$. $F = 11001001011000011010$ ($|F| = 20$ bits) is divided into $m = 5$ blocks: $\bar{\mathbf{w}}_0 = 1100, \bar{\mathbf{w}}_1 = 1001, \bar{\mathbf{w}}_2 = 0110, \bar{\mathbf{w}}_3 = 0001$ and $\bar{\mathbf{w}}_4 = 1010$ ($|\bar{\mathbf{w}}_i| = 4$). Suppose $n = 8$, we make 8 coded block $\{\mathbf{c}_0, \cdots, \mathbf{c}_7\}$. To make $\mathbf{c}_0$, $v_0 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 = 0011$ is used (as explained in (*)). Since the number of bit '1' in v_0 is 2, we have $\hat{\mathbf{c}}_0 = 2$. Since $\hat{\mathbf{c}}_0 = 2$ and $|\bar{\mathbf{w}}_i| = 4$, we have $P_0 = \{0011, 0101, 0110, 1001, 1010, 1100\}$ in which the elements are sorted in an ascending order and indexed as $\{0, 1, \cdots, 5\}$. The important thing is that: since $|P_0| = \binom{4}{2} = 6$, we only need $\log_2 6 \simeq 3$ bits to represent $\mathbf{c}_i$ instead of 4 bits of $\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2$. Since the index of $v_0 = 0011$ in P_0 is 0, we have $\mathbf{c}_0 = 0_{(decimal)} = 000$. $\{\mathbf{c}_0, \hat{\mathbf{c}}_0\}$ are sent to the server $\mathcal{S}_0$. Similarly, we make $\mathbf{c}_1, \cdots \mathbf{c}_7$. The results are as follows:

- $v_0 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 = 0011$, $\hat{\mathbf{c}}_0 = 2$, $P_0 = \{0011, 0101, 0110, 1001, 1010, 1100\}$, $\mathbf{c}_0 = 0_{dec} = 000$.
- $v_1 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_3 = 0100$, $\hat{\mathbf{c}}_1 = 1$, $P_1 = \{0001, 0010, 0100, 1000\}$, $\mathbf{c}_1 = 2_{dec} = 10$.
- $v_2 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_4 = 1111$, $\hat{\mathbf{c}}_2 = 4$, $P_2 = \{1111\}$, $\mathbf{c}_2 = 0_{dec} = 0$.
- $v_3 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_3 = 1011$, $\hat{\mathbf{c}}_3 = 3$, $P_3 = \{0111, 1011, 1101, 1110\}$, $\mathbf{c}_3 = 1_{dec} = 01$.

- $v_4 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_4 = 0000$, $\hat{\mathbf{c}}_4 = 0$, $P_4 = \{\}$, $\mathbf{c}_4 = 0_{dec} = 0$.
- $v_5 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_3 \oplus \bar{\mathbf{w}}_4 = 0111$, $\hat{\mathbf{c}}_5 = 3$, $P_5 = \{0111, 1011, 1101, 1110\}$, $\mathbf{c}_5 = 0_{dec} = 00$.
- $v_6 = \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_3 = 1110$, $\hat{\mathbf{c}}_6 = 3$, $P_6 = \{0111, 1011, 1101, 1110\}$, $\mathbf{c}_6 = 3_{dec} = 11$.
- $v_7 = \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_4 = 0101$, $\hat{\mathbf{c}}_7 = 2$, $P_7 = \{0011, 0101, 0110, 1001, 1010, 1100\}$, $\mathbf{c}_7 = 1_{dec} = 001$.

B. Retrieve

TABLE II: Retrieve algorithm (Retrieve)

INPUT: $\{\mathbf{c}_{k_0}, \hat{\mathbf{c}}_{k_0}\}, \{\mathbf{c}_{k_1}, \hat{\mathbf{c}}_{k_1}\}, \cdots, \{\mathbf{c}_{k_{m-1}}, \hat{\mathbf{c}}_{k_{m-1}}\}$
OUTPUT: $\{\bar{\mathbf{w}}_0, \bar{\mathbf{w}}_1, \cdots, \bar{\mathbf{w}}_{m-1}\}$

1: $i \leftarrow 0$
2: **for** $i \leftarrow 0$ **to** $i \leftarrow m-1$
3: $\quad v_{k_i} \leftarrow SUB(\mathbf{c}_{k_i}, \hat{\mathbf{c}}_{k_i}, \frac{|F|}{m})$
4: $\quad \alpha_{k_i}, \beta_{k_i}, \gamma_{k_i} \leftarrow GET(m, k_i)$
5: $\quad u_{k_i} \leftarrow COEF(v_{k_i}, m, \alpha_{k_i}, \beta_{k_i}, \gamma_{k_i})$
6: $\bar{U} \leftarrow [u_{k_0}, u_{k_1}, \cdots, u_{k_{m-1}}]^T$
7: $U = gaussian\_elimination(\bar{U})$
8: $\{\bar{\mathbf{w}}_0, \bar{\mathbf{w}}_1, \cdots, \bar{\mathbf{w}}_{m-1}\} \leftarrow filter(U)$
9: $F \leftarrow \bar{\mathbf{w}}_0 || \bar{\mathbf{w}}_1 || \cdots || \bar{\mathbf{w}}_{m-1}$
return F

To retrieve F, $\mathcal{C}$ requires m servers to provide their coded blocks (suppose $\mathbf{c}_{k_0}$, $\cdots$, $\mathbf{c}_{k_{m-1}}$). *Note that the coded blocks are chosen such that the binary matrix consisting of coefficient vectors of the XORs should have full rank.*

The main algorithm Retrieve is described in Table II. On input m sets, each set consists of: coded block $\mathbf{c}_{k_i}$ and metadata $\hat{\mathbf{c}}_{k_i}$, this algorithm outputs m file blocks $\{\bar{\mathbf{w}}_0, \cdots, \bar{\mathbf{w}}_{m-1}\}$. SUB, GET and $COEF$ are the sub-algorithms. Firstly, for each coded block $\mathbf{c}_{k_i}$, $\mathcal{C}$ finds the XOR used to construct $\mathbf{c}_{k_i}$ (line 3: SUB denotes the function used to find the XOR on input $\mathbf{c}_{k_i}$, $\hat{\mathbf{c}}_{k_i}$ and the length of a file block $\frac{|F|}{m}$, then returning result to v_{k_i}). $\mathcal{C}$ then performs the GET algorithm to find the indices of three operands of the XOR (line 4). $\mathcal{C}$ constructs a vector consisting of $m+1$ elements: $u_{k_i} = (e_0, e_2, \cdots, e_{m-1}, v_{k_i})$ where $e_i \in \{0, 1\}$ for $i = 0, \cdots, m-1$. $e_i = 1$ where i is the first-operand index (α_{k_i}), the second operand index (β_{k_i}) and the third operand index (γ_{k_i}) of the XOR. $e_i = 0$ elsewhere. u_{k_i} is constructed by the $COEF$ algorithm (line 5). Secondly, all u_{k_i} where $i = [0, m-1]$ are combined into a matrix $\bar{U}$ (line 6). Thirdly, $\mathcal{C}$ executes Gaussian elimination *gaussian_elimination* on $\bar{U}$ and returns to a matrix U (line 7) to solve the equation system of m variables $\bar{\mathbf{w}}_0, \cdots, \bar{\mathbf{w}}_{m-1}$. Fourthly, $\mathcal{C}$ filters $\bar{\mathbf{w}}_0, \cdots, \bar{\mathbf{w}}_{m-1}$ from U (line 8). Finally, F is reconstructed as $\bar{\mathbf{w}}_0 || \cdots || \bar{\mathbf{w}}_{m-1}$ (line 9).

TABLE III: SUB algorithm

Input: $\mathbf{c}_i, \hat{\mathbf{c}}_i, |F|, m$
Output: v_i

$P_i \leftarrow list\_all\_combinations(\frac{|F|}{m}, \hat{\mathbf{c}}_i)$
$count \leftarrow 0$
for $x \leftarrow 0$ **to** $P_i.length - 1$
$\quad$ **if** $(count == {}_i)$
$\quad\quad$ **return** $P_i[x]$
return $P_i[count]$

TABLE IV: GET algorithm

Input: m, k_i
Output: $\alpha_{k_i}, \beta_{k_i}, \gamma_{k_i}$

$count = -1$
for $\alpha \leftarrow 0$ **to** $m-3$
$\quad$ **for** $\beta \leftarrow \alpha + 1$ **to** $m-2$
$\quad\quad$ **for** $\gamma \leftarrow \beta + 1$ **to** $m-1$
$\quad\quad\quad count++$
$\quad\quad\quad$ **if** $(count == k_i)$
$\quad\quad\quad\quad \alpha_{k_i} \leftarrow \alpha$
$\quad\quad\quad\quad \beta_{k_i} \leftarrow \beta$
$\quad\quad\quad\quad \gamma_{k_i} \leftarrow \gamma$
return $\alpha_{k_i}, \beta_{k_i}, \gamma_{k_i}$

TABLE V: COEF algorithm

Input: $v_i, m, \alpha_i, \beta_i, \gamma_i$
Output: u_i

for $x \leftarrow 0$ **to** $x \leftarrow m-1$
$\quad$ **if** $((x == \alpha_i)$ **or** $(x == \beta_i)$ **or** $(x == \gamma_i))$
$\quad\quad u_i[x] \leftarrow 1$
$\quad$ **else** $u_i[x] \leftarrow 0$
$u_i[m] \leftarrow v_i$
return u_i

Example 2: We reconstruct F from Example 1. Assume that $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$ and $\mathbf{c}_6$ are chosen to reconstruct F since the matrix consisting of coefficient vectors of v_0, v_1, v_2, v_3 and v_6 has full rank. Since $\mathbf{c}_0 = 000, \hat{\mathbf{c}}_0 = 2$, we have $P_0 = \{0011, 0101, 0110, 1001, 1010, 1100\}$, map $\mathbf{c}_0$ to obtain $v_0 = 0011$. Similarly, we have $v_1 = 0100$, $v_2 = 1111$, $v_3 = 1011$ and $v_6 = 1110$. Next, a vector u_{k_i} is constructed for each $\mathbf{c}_{k_i}$. $(\alpha_0 = 0, \beta_0 = 1, \gamma_0 = 2)$ because $\mathbf{c}_0$ uses $\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2$. Given $\alpha_0 = 0, \beta_0 = 1$, $\gamma_0 = 2$ and $v_0 = 0011$, we have $u_0 = [1, 1, 1, 0, 0, 0011]$. Similarly, we have $u_1 = [1, 1, 0, 1, 0, 0100]$, $u_2 = [1, 1, 0, 0, 1, 1111]$, $u_3 = [1, 0, 1, 1, 0, 1011]$ and $u_6 = [0, 1, 1, 1, 0, 1110]$. From u_0, u_1, u_2, u_3, u_6, we construct $\bar{U}$ as:

$$\bar{U} = \begin{pmatrix} 1,1,1,0,0,0011 \\ 1,1,0,1,0,0100 \\ 1,1,0,0,1,1111 \\ 1,0,1,1,0,1011 \\ 0,1,1,1,0,1110 \end{pmatrix}$$

Applying Gaussian elimination on $\bar{U}$, we have U as below.

$$U = \begin{pmatrix} 1,0,0,0,0,1100 \\ 0,1,0,0,0,1001 \\ 0,0,1,0,0,0110 \\ 0,0,0,1,0,0001 \\ 0,0,0,0,1,1010 \end{pmatrix}$$

From U, we obtain $\bar{\mathbf{w}}_0 = 1100, \bar{\mathbf{w}}_1 = 1001, \bar{\mathbf{w}}_2 = 0110, \bar{\mathbf{w}}_3 = 0001$ and $\bar{\mathbf{w}}_4 = 1010$. Finally, F is reconstructed as $F = \bar{\mathbf{w}}_0 || \bar{\mathbf{w}}_1 || \bar{\mathbf{w}}_2 || \bar{\mathbf{w}}_3 || \bar{\mathbf{w}}_4$.

C. Repair

The main algorithm Repair is described in Table VI. GET, $REGET$ and SUB are the sub-algorithms. Assume that $\mathcal{S}_y$ is

TABLE VI: Repair algorithm (Repair)

INPUT: $\mathcal{S}_y$
OUTPUT: $\mathbf{c}_y, \hat{\mathbf{c}}_y$
01: $\alpha_y, \beta_y, \gamma_y \leftarrow GET(m, y)$
02: choose $a, b \in \{0, m-1\}$ s.t. $a, b \neq \alpha_y, \beta_y, \gamma_y$
03: $\{\alpha_{r_1}, \beta_{r_1}, \gamma_{r_1}\} \leftarrow ascending\_sort(\alpha_y, \beta_y, a)$
04: $\{\alpha_{r_2}, \beta_{r_2}, \gamma_{r_2}\} \leftarrow ascending\_sort(\alpha_y, \gamma_y, b)$
05: $\{\alpha_{r_3}, \beta_{r_3}, z\gamma_{r_3}\} \leftarrow ascending\_sort(\alpha_y, a, b)$
06: $\mathcal{S}_{r_1} \leftarrow REGET(\alpha_{r_1}, \beta_{r_1}, \gamma_{r_1})$
07: $\mathcal{S}_{r_2} \leftarrow REGET(\alpha_{r_2}, \beta_{r_2}, \gamma_{r_2})$
08: $\mathcal{S}_{r_3} \leftarrow REGET(\alpha_{r_3}, \beta_{r_3}, \gamma_{r_3})$
09: Require $\mathcal{S}_{r_1}, \mathcal{S}_{r_2}, \mathcal{S}_{r_3}$ to provide $\{\mathbf{c}_{r_1}, \hat{\mathbf{c}}_{r_1}\}, \{\mathbf{c}_{r_2}, \hat{\mathbf{c}}_{r_2}\}, \{\mathbf{c}_{r_3}, \hat{\mathbf{c}}_{r_3}\}$
10: $v_{r_1} \leftarrow SUB(\mathbf{c}_{r_1}, \hat{\mathbf{c}}_{r_1}, \frac{\lvert F\rvert}{m})$
11: $v_{r_2} \leftarrow SUB(\mathbf{c}_{r_2}, \hat{\mathbf{c}}_{r_2}, \frac{\lvert F\rvert}{m})$
12: $v_{r_3} \leftarrow SUB(\mathbf{c}_{r_3}, \hat{\mathbf{c}}_{r_3}, \frac{\lvert F\rvert}{m})$
13: $v_y \leftarrow v_{r_1} \oplus v_{r_2} \oplus v_{r_3}$
14: $\hat{\mathbf{c}}_y \leftarrow number\_bit1(v_y)$
15: $P_y \leftarrow list\_all\_combinations(\frac{\lvert F\rvert}{m}, \hat{\mathbf{c}}_y)$
16: $\mathbf{c}_y \leftarrow ord(P_y, v_y)$
return $\mathbf{c}_y, \hat{\mathbf{c}}_y$

TABLE VII: REGET algorithm

Input: $\alpha_{r_i}, \beta_{r_i}, \gamma_{r_i}$
Output: $\mathcal{S}_{r_i}$
1: $count \leftarrow -1$
2: **for** $\alpha \leftarrow 0$ **to** $m-3$
3: **for** $\beta \leftarrow \alpha + 1$ **to** $m-2$
4: **for** $\gamma \leftarrow \beta + 1$ **to** $m-1$
4: $count++$
5: **if** $(\alpha == \alpha_{r_i})$ **and** $(\beta == \beta_{r_i})$ **and** $(\gamma == \gamma_{r_i})$
6: **return** $count$

corrupted, it is repaired using three healthy servers. Firstly, $\mathcal{C}$ finds the indices of three operands of the XOR $\bar{\mathbf{w}}_{\alpha_y} \oplus \bar{\mathbf{w}}_{\beta_y} \oplus \bar{\mathbf{w}}_{\gamma_y}$ in $\mathcal{S}_y$ by using the GET algorithm (line 1). Secondly, let a, b be two numbers in $\{0, m-1\}$ such that $a, b \neq \alpha_y, \beta_y, \gamma_y$ (line 2). The idea to find such three coded blocks to repair $\mathcal{S}_y$ is that:

$$\begin{aligned} \bar{\mathbf{w}}_{\alpha_y} \oplus \bar{\mathbf{w}}_{\beta_y} \oplus \bar{\mathbf{w}}_{\gamma_y} &= (\bar{\mathbf{w}}_{\alpha_y} \oplus \bar{\mathbf{w}}_{\beta_y} \oplus \bar{\mathbf{w}}_a) \\ &\oplus (\bar{\mathbf{w}}_{\alpha_y} \oplus \bar{\mathbf{w}}_{\gamma_y} \oplus \bar{\mathbf{w}}_b) \\ &\oplus (\bar{\mathbf{w}}_{\alpha_y} \oplus \bar{\mathbf{w}}_a \oplus \bar{\mathbf{w}}_b) \end{aligned}$$

Thirdly, $\{\alpha_y, \beta_y, a\}$, $\{\alpha_y, \gamma_y, b\}$ and $\{\alpha_y, a, b\}$ are sorted in ascending orders using the $ascending\_sort$ algorithm (line 3-5). Let $\{\alpha_{r_1}, \beta_{r_1}, \gamma_{r_1}\}$, $\{\alpha_{r_2}, \beta_{r_2}, \gamma_{r_2}\}$ and $\{\alpha_{r_3}, \beta_{r_3}, \gamma_{r_3}\}$ be the results of these sorting, respectively. The explanation of $ascending\_sort$ is skipped since it is a simple programming function. Fourthly, $\mathcal{C}$ finds three servers storing three XORs by using the $REGET$ algorithm (line 6-8). Fifthly, $\mathcal{C}$ finds the real result of XORs $(v_{r_1}, v_{r_2}, v_{r_3})$ by using the SUB algorithm (line 10-12). Sixthly, the XOR of $\mathcal{S}_y$ is recovered (line 13) by $v_y = v_{r_1} \oplus v_{r_2} \oplus v_{r_3}$. $\mathcal{C}$ then finds the metadata $\hat{\mathbf{c}}_y$ of $\mathbf{c}_y$ by counting the number of bits $'1'$ in v_y (line 14). Finally, we find the coded block of $\mathcal{S}_y$ (line 15-16) like the encode phase.

Example 3 We follow example 1, 2. Assume $\mathcal{S}_3$ is corrupted. $\alpha_y = 0, \beta_y = 2$ and $\gamma_y = 3$ because $\mathcal{S}_3$ uses $\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_3$ (line 1). $a = 1$ and $b = 4$ are chosen because $1, 4 \neq 0, 2, 3$ (line 2). Observe that:

$$\begin{aligned} \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_3 &= (\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_1) \\ &\oplus (\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_3 \oplus \bar{\mathbf{w}}_4) \\ &\oplus (\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_4) \end{aligned}$$

Then, $\{0, 2, 1\}$, $\{0, 3, 4\}$ and $\{0, 1, 4\}$ are sorted in ascending orders as $\{0, 1, 2\}$, $\{0, 3, 4\}$ and $\{0, 1, 4\}$. Let $\{\alpha_{r_1}, \beta_{r_1}, \gamma_{r_1}\} = \{0, 1, 2\}$, $\{\alpha_{r_2}, \beta_{r_2}, \gamma_{r_2}\} = \{0, 3, 4\}$ and $\{\alpha_{r_3}, \beta_{r_3}, \gamma_{r_3}\} = \{0, 1, 4\}$ (line 3-5). Given $\{\alpha_{r_1}, \beta_{r_1}, \gamma_{r_1}\} = \{0, 1, 2\}$, we find $\mathcal{S}_0$ because $\mathcal{S}_0$ uses $\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2$. Given $\{\alpha_{r_2}, \beta_{r_2}, \gamma_{r_2}\} = \{0, 3, 4\}$, we find $\mathcal{S}_5$ because $\mathcal{S}_5$ uses $\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_3 \oplus \bar{\mathbf{w}}_4$. Given $\{\alpha_{r_3}, \beta_{r_3}, \gamma_{r_3}\} = \{0, 1, 4\}$, we find $\mathcal{S}_2$ because $\mathcal{S}_2$ uses $\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_4$ (line 6-8). $\mathcal{S}_0, \mathcal{S}_5$ and $\mathcal{S}_2$ are required to provide $\{\mathbf{c}_0, \hat{\mathbf{c}}_0\}, \{\mathbf{c}_5, \hat{\mathbf{c}}_5\}$ and $\{\mathbf{c}_2, \hat{\mathbf{c}}_2\}$ (line 9). Given $\{\mathbf{c}_0 = 000, \hat{\mathbf{c}}_0 = 2\}$, we find $v_0 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 = 0011$. Given $\{\mathbf{c}_5 = 00, \hat{\mathbf{c}}_5 = 3\}$, we find $v_5 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_3 \oplus \bar{\mathbf{w}}_4 = 0111$. Given $\{\mathbf{c}_2 = 0, \hat{\mathbf{c}}_2 = 4\}$, we find $v_2 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_4 = 1111$ (line 10-12). Then, $v_y = v_0 \oplus v_5 \oplus v_2 = 1011$ (line 13). Then, $\hat{\mathbf{c}}_y = 3$ since the number of bits '1' of 1011 is 3. Given $\hat{\mathbf{c}}_y = 3$ and $\lvert\bar{\mathbf{w}}_i\rvert = 4$, $P_y = \{0111, 1011, 1101, 1110\}$. Since $\lvert P_y\rvert = 4$, we need $\log_2 4 = 2$ bits to present $\mathbf{c}_y$. The coded block $\mathbf{c}_y$ is the index of v_y in P_y which is: $\mathbf{c}_y = 1_{decimal} = 01$.

Unlike network coding, our new coded block is exactly same as the corrupted coded block.

IV. Security analysis

A. Data recover condition

Let *epoch* be a time step in which the servers are checked. If a corrupted server is detected, it is repaired in the next epoch.

Theorem 1: *The raw file F can be retrieved as long as in any epoch, at least m out of n servers are healthy and the corresponding matrix has full rank, i.e., rank equals to m.*

Proof: F has m blocks: $F = \bar{\mathbf{w}}_0 || \cdots || \bar{\mathbf{w}}_{m-1}$, the number of coded blocks is n. Each coded block $\mathbf{c}_i$ is computed using XOR between three different file blocks: $v_i = \bar{\mathbf{w}}_\alpha \oplus \bar{\mathbf{w}}_\beta \oplus \bar{\mathbf{w}}_\gamma$. To retrieve F, we view $\bar{\mathbf{w}}_0, \cdots, \bar{\mathbf{w}}_{m-1}$ as the variables that need to be solved. To solve these m variables, we need at least m coded blocks which make the matrix have full rank because the number of variables in a equation system has to be less than or equal to the number of equations.

$$\begin{cases} v_0 = \bar{\mathbf{w}}_{\alpha_0} \oplus \bar{\mathbf{w}}_{\beta_0} \oplus \bar{\mathbf{w}}_{\gamma_0} \\ v_1 = \bar{\mathbf{w}}_{\alpha_1} \oplus \bar{\mathbf{w}}_{\beta_1} \oplus \bar{\mathbf{w}}_{\gamma_1} \\ \ldots \\ v_{m-1} = \bar{\mathbf{w}}_{\alpha_{m-1}} \oplus \bar{\mathbf{w}}_{\beta_{m-1}} \oplus \bar{\mathbf{w}}_{\gamma_{m-1}} \end{cases}$$

The number of required servers is thus at least m in any epoch. ■

B. Security threshold

In this subsection, we show the security of SW-POR using entropy theory. Let $H(x)$ be the Shannon's entropy of a random variable x. Let λ be the number of the pairs of coded block and its metadata that an adversary $\mathcal{A}$ can learn. Let L be the number of servers whose coded blocks are constructed collectively from m file blocks $\bar{\mathbf{w}}_0, \cdots, \bar{\mathbf{w}}_{m-1}$ via XOR.

Theorem 2: *The advantage of the adversary $\mathcal{A}$ who has λ pairs of coded blocks and metadata is as follows:*

TABLE VIII: Comparison between network coding and SW-POR

		Network coding	**SW-POR**
Feature	Exact-repair	No	Yes
Storage	All phases	$O(m+\|\bar{\mathbf{w}}\|)$	$O(\log_2\binom{\|\bar{\mathbf{w}}\|}{\hat{\mathbf{c}}}+\log_2\|\bar{\mathbf{w}}\|)$
Computation complexity	Encode	$O(mn)$	$O(n)$
	Retrieve	$O(m)$	$O(m)$
	Repair	$O(m)$	$O(1)$
Communication complexity	Encode	$O(n(m+\|\bar{\mathbf{w}}\|))$	$O(n(\log_2\binom{\|\bar{\mathbf{w}}\|}{\hat{\mathbf{c}}}+\log_2\|\bar{\mathbf{w}}\|))$
	Retrieve	$O(m(m+\|\bar{\mathbf{w}}\|))$	$O(m(\log_2\binom{\|\bar{\mathbf{w}}\|}{\hat{\mathbf{c}}}+\log_2\|\bar{\mathbf{w}}\|))$
	Repair	$O((m+1)(m+\|\bar{\mathbf{w}}\|))$	$O(3(\log_2\binom{\|\bar{\mathbf{w}}\|}{\hat{\mathbf{c}}}+\log_2\|\bar{\mathbf{w}}\|))$

$$\mathsf{Pr}_{\mathcal{A}}[(\mathbf{c}_{i_0},\hat{\mathbf{c}}_{i_0}),\cdots,(\mathbf{c}_{i_{\lambda-1}},\hat{\mathbf{c}}_{i_{\lambda-1}})] = \begin{cases} H(F)(\lambda<L) \\ \frac{m-\lambda}{m}H(F)(L\le\lambda<m) \\ 0(m\le\lambda) \end{cases}$$

Proof: The probability for $\mathcal{A}$ to obtain F is the entropy:

$$\mathsf{Pr}_{\mathcal{A}}[(\mathbf{c}_{i_0},\hat{\mathbf{c}}_{i_0}),...,(\mathbf{c}_{i_{\lambda-1}},\hat{\mathbf{c}}_{i_{\lambda-1}})] = H(F|(C_{i_0},\hat{C}_{i_0}),\cdots,(C_{i_{\lambda-1}},\hat{C}_{i_{\lambda-1}}))$$

where $C_{i_1},\cdots,C_{i_\lambda}$ are the random variables of λ coded blocks $\mathbf{c}_{i_0},\cdots\mathbf{c}_{i_{\lambda-1}}$, and $\hat{C}_{i_1},\cdots,\hat{C}_{i_\lambda}$ are the random variables of λ corresponding metadata $\hat{\mathbf{c}}_{i_0},\cdots\hat{\mathbf{c}}_{i_{\lambda-1}}$. From the property of conditional entropy, we have:

$$H(F|(C_{i_0},\hat{C}_{i_0}),\cdots,(C_{i_{\lambda-1}},\hat{C}_{i_{\lambda-1}}))\le H(F|v_{i_0},\cdots,v_{i_{\lambda-1}})$$

where v_i denotes the XOR that is uniquely determined by a coded block $\mathbf{c}_i$ and its metadata $\hat{\mathbf{c}}_i$. The equality is halt if F is uniformly distributed.

When $\lambda<L$, the set $\{v_{i_0},\cdots,v_{i_{\lambda-1}}\}$ are constructed from inadequate m file blocks; and the binary matrix consisting of coefficients vectors of v_{i_j} has not full rank, we have: $H(F|v_{i_0},\cdots,v_{i_{\lambda-1}})=H(F)$. This yields:

$$H(F|(C_{i_0},\hat{C}_{i_0}),\cdots,(C_{i_{\lambda-1}},\hat{C}_{i_{\lambda-1}}))=H(F)\quad(\lambda<L)$$

When $L\le\lambda$, the binary matrix consisting of coefficients vectors of v_{i_j} has full rank λ, we have: $H(F|v_{i_0},\cdots,c_{i_{\lambda-1}})=\frac{m-\lambda}{\lambda}H(F)$. This yields:

$$H(F|(C_{i_0},\hat{C}_{i_0}),\cdots,(C_{i_{m-1}},\hat{C}_{i_{\lambda-1}}))=\frac{m-\lambda}{\lambda}H(F)< H(F)\quad(L\le\lambda<m)$$

When $m\le\lambda$, from (2) we can obtain $\frac{m-\lambda}{\lambda}H(F)=0$. This yields:

$$H(F|(C_{i_0},\hat{C}_{i_0}),\cdots,(C_{i_{\lambda-1}},\hat{C}_{i_{\lambda-1}}))=0\quad(m\le\lambda)$$

■

V. Efficiency analysis

A. Storage cost

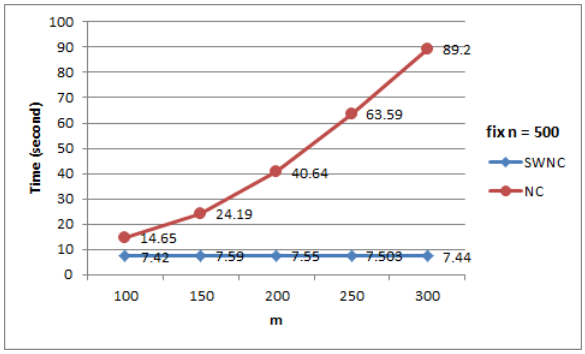
In network coding, since the size of a coded block is $m+|\bar{\mathbf{w}}|$, the storage cost in each server is $O(m+|\bar{\mathbf{w}}|)$. In SW-POR, each server store two things: $\mathbf{c}_i$ which has the size $|\mathbf{c}_i|=\log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i}$ and $\hat{\mathbf{c}}_i$ which has the size $|\hat{\mathbf{c}}_i|=\log_2|\bar{\mathbf{w}}|$ because $\hat{\mathbf{c}}_i\in\{1,|\bar{\mathbf{w}}|\}$. Our storage cost is thus: $O(\log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i}+\log_2|\bar{\mathbf{w}}|)$. For all $|\bar{\mathbf{w}}|$ and $\hat{\mathbf{c}}_i\in\{0,|\bar{\mathbf{w}}|\}$, $\log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i}<|\bar{\mathbf{w}}|$. Therefore, $(\log_2\binom{|\bar{\mathbf{w}}|}{d}+\log_2|\bar{\mathbf{w}}|)<(m+|\bar{\mathbf{w}}|)$ if $\log_2|\bar{\mathbf{w}}|<m$. The condition is halt if the parameters are chosen s.t. $|\bar{\mathbf{w}}|<2^m$.

B. Communication cost

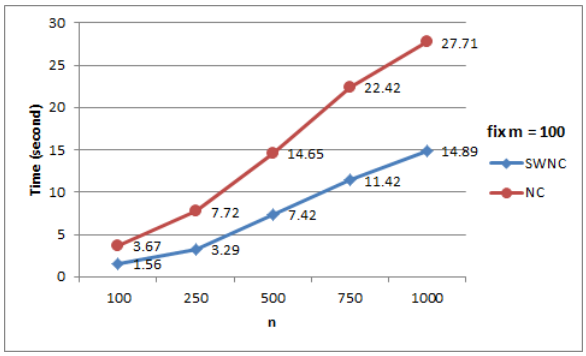
1) Encode: In network coding, the size of each packet is $m+|\bar{\mathbf{w}}|$; thus, the cost for $\mathcal{C}$ to send n packets to n servers is $O(n(m+|\bar{\mathbf{w}}|))$. In SW-POR, the size of each packet is $\log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i}+\log_2|\bar{\mathbf{w}}|$; thus, the cost for $\mathcal{C}$ sends n packets to n servers is $O(n(\log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i}+\log_2|\bar{\mathbf{w}}|))$.

2) Retrieve: Suppose $\mathcal{C}$ uses m out of n servers to retrieve F. In network coding, the cost for m servers to send their packets to $\mathcal{C}$ is $O(m(m+|\bar{\mathbf{w}}|))$. Similarly, the cost in SW-POR is $O(m(\log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i}+\log_2|\bar{\mathbf{w}}|))$.

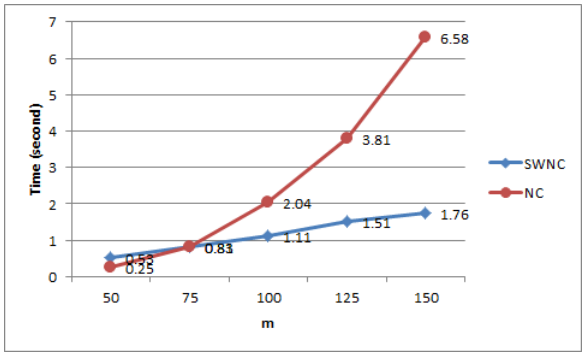
3) Repair: In network coding, the cost for m healthy servers to provide their packets to $\mathcal{C}$ is $m(m+|\bar{\mathbf{w}}|)$ and the cost for $\mathcal{C}$ to send new coded blocks to the new server is $m+|\bar{\mathbf{w}}|$. Thus, the cost is $O((m+1)(m+|\bar{\mathbf{w}}|))$. In SW-POR, we only need three healthy servers to repair. Our cost is thus $O(3(\log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i}+\log_2|\bar{\mathbf{w}}|))$.



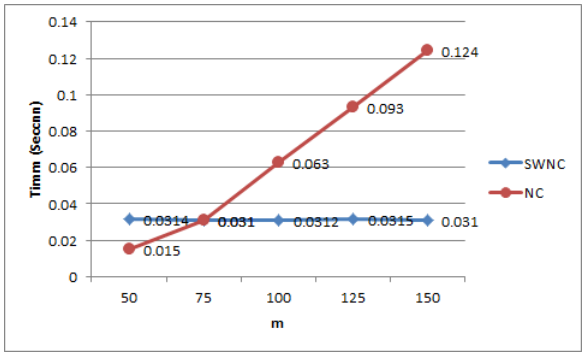
(a) Encoding phase: fix n



(b) Encoding phase: fix m



(c) Retrieving phase



(d) Repairing phase

Fig. 2: Comparison of the computational overhead between network coding and SW-POR

C. Computation cost

1) Encode: In network coding, to compute a coded block, $\mathcal{C}$ combines m augmented blocks that takes $O(m)$ operations. Thus, $\mathcal{C}$ needs $O(mn)$ operations to make n coded blocks. In SW-POR, to make a coded block, $\mathcal{C}$ only computes XOR between three file blocks that takes two operations. Thus, $\mathcal{C}$ needs $O(n)$ to make n coded blocks.

2) Retrieve: In network coding, $\mathcal{C}$ requires m healthy coded blocks that takes $O(m)$ operations. Similarly, in SW-POR, $\mathcal{C}$ also needs $O(m)$ operations.

3) Repair: In network coding, $\mathcal{C}$ needs at least m coded blocks from m healthy servers for repairing. This has $O(m)$ operations. In SW-POR, we only need three healthy coded blocks for repairing. This has $O(1)$ operations.

VI. Experiment

We now assess the performance of network coding and the proposed scheme. We implement by Python using a computer with Intel Core i5 processor running at 2.40GHz, 4.00GB of RAM, Win 7 64-bit OS. For network coding scheme, we use $|q|$ as 2^{10} bits. For both network coding and SW-POR, we set each block as 2^{10} bits and each server stores one coded block. The results are the average of 100 runs. In both network coding and SW-POR implementations, gmpy2 library is used for dealing with big values, itertools library is used for efficient looping of big values. Figure 2 compares the computation overheads between network coding and SW-POR. Graph 2a compares the encode phase when fixing $n = 500$. Graph 2b compares the encode phase when fixing $m = 100$. Graph 2c compares the retrieve phase and Graph 2d compares the repair phase.

Graph 2a and 2d show that the time-consuming of network coding increases linearly with m while that of SW-POR is almost constant. Graph 2b and 2c show that the time-consumings in both network coding and SW-POR increase linearly with n and m, respectively, but the slope of SW-POR is less than that of network coding. The experiment results are almost match with the complexity analysis in Section V.

VII. Conclusion

In this paper, we propose a new POR scheme named SW-POR to support exact-repair and to obtain an optimal coded block size. We also achieve better storage, communication and computation costs. Our idea is based on the binning index approach of Slepian-Wolf code which is a common data compression code in network. We show the security threshold of SW-POR based on entropy theory. We analyse the efficiency based on complexity theory and implement our scheme and network-coding POR.

References

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

PCs", in *Proc. of ACM conf. on Measurement and modeling of computer systems - SIGMETRICS'00*, 2000, pp.34-43.

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

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

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

[5] S. Li, R. Yeung, and N. Cai, "Linear Network Coding", in *IEEE Trans. on Information Theory*, vol.49, no.2, pp.371381, 2003.

[6] Ralf Koetter and Muriel Mdard, "An Algebraic Approach to Network Coding", in *IEEE/ACM Trans. on Networking (TON)*, vol.11, no.5, Oct 2003, pp.782-795.

[7] A. Juels and B.Kaliski, "PORs: Proofs of retrievability for large files", in *Proc. 14th ACM Computer and communications security Conf.*, 2007, pp. 584-597.

[8] H. Shacham and B. Waters, "Compact Proofs of Retrievability", in *Proc. 14th Int. Conf. on the Theory and Application of Cryptology and Information Security: Advances in Cryptology - ASIACRYPT'08*, Dec 2008, pp.90-107.

[9] K. Bowers, A. Juels and A. Oprea, "Proofs of retrievability: theory and implementation", in *Proc. ACM workshop on cloud computing security*, 2009, pp.43-54.

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

[11] J. Li, S. Yang, X. Wang, X. Xue and B. Li, "Tree-structured Data Regeneration in Distributed Storage Systems with Network Coding", in *Proc. 29th IEEE Information communications Conf.*, 2000, pp.2892-2900.

[12] B. Chen, R. Curtmola, G. Ateniese and R. Burns, "Remote Data Checking for Network Coding-based Distributed Storage Systems", in *Proc. ACM workshop on cloud computing security*, 2010, pp. 31-42.

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

[14] K. Bowers, A. Juels and A. Oprea, "HAIL, A high-availability and integrity layer for cloud Storage", in *Proc. 16th ACM Computer and communications security Conf.*, 2009, pp. 187-198.

[15] Y. Dodis, S. Vadhan and D. Wichs, "Proofs of Retrievability via Hardness Amplification", in *Proc. 6th Theory of Cryptography Conference on Theory of Cryptography - TCC'09*, Mar 2009, pp.109-127

[16] J. Hendricks, G. R. Ganger and M. Reiter, "Verifying distributed erasure-coded data", in *Proc. 26th ACM Principles of Distributed Computing Symposium*, 2007, pp.163-168.

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

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

[19] C. Cheng, T. Jiang and Qian Zhang, "TESLA-Based Homomorphic MAC for Authentication in P2P System for Live Streaming with Network Coding", in *IEEE Journal on Selected Areas in Communications*, vol.31, no.9, Sep 2013, pp.291-298.

[20] Z.Zhang, Q. Lian, S. Lin, W.Chen, Y.Chen and C. Jin, "Bitvault: A highly reliable distributed data retention platform", in *ACM SIGOPS Operating Systems Review*, vol.41, Apr 2007, pp.27-36.

[21] Q. Wang, C. Wang, K. Ren, W. Lou and J. Li, "Enabling Public Auditability and Data Dynamics for Storage Security in cloud Computing", in *IEEE Trans. parallel and distributed system*, vol.22, no.5, May 2011, pp.847-859.

[22] D. Catalano, D. Fiore and B. Warinschi, "Efficient network coding signature in the standard model", in *Proc. 15th Practice and Theory in Public Key Cryptography Conf.*, 2012, pp.680-696.

[23] W. Yana, M. Yanga, L. Lia and H. Fang, "Short signature scheme for multi-source network coding", in *Journal Computer Communications*, vol.35, no.3, Feb 2012, pp.344-351.

[24] Samuel Cheng, *Slepian-Wolf Code Designs*, 2010, Available: http://tulsagrad.ou.edu/samuel_cheng/information_theory_2010/swcd.pdf.

[25] V. Stankovic, A.D. Liveris, Z. Xiong and C.N. Georghiades, "On code design for the Slepian-Wolf problem and lossless multiterminal networks", in *IEEE Trans. on Information Theory*, vol.52, no.4, Apr 2006, pp.1495-1507.

[26] V. Stankovi, A.D. Liveris, Z. Xiong and C.N.Georghiades, "Design of Slepian-Wolf Codes by Channel Code Partitioning", in *Proc. of Data Compression Conf. - DCC'04*, Mar 2004, pp.302-311.

[27] R. Johnson, D. Molnar, D. Song and D Wagner, "Homomorphic Signature Schemes", in *Proc. of Cryptographer's Track at the RSA Conf. on Topics in Cryptology - CT-RSA'02*, pp.244-262, 2002.

[28] N. Attrapadung and B. Libert, "Homomorphic network coding signatures in the standard model", in *Proc. of 14th Int. conf. on Practice and theory in public key cryptography conference on Public key cryptography - PKC*, Mar 2011, pp.680-696.

[29] David Mandell Freeman, "Improved security for linearly homomorphic signatures: a generic framework", in *Proc. of 15th conf. on Practice and Theory in Public Key Cryptography - PKC 2012*, vol.7293, May 2012, pp.697-714.