SW-POR: A Novel POR Scheme using Slepian-Wolf Coding 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 the clients can outsource their data to the cloud storage server to deal with the local storage limitation. However, the cloud storage providers are untrustworthy, which can lead to several security challenges, such as data availability, data integrity, and data confidentiality. To mitigate the issues of data availability and data integrity, a novel Slepian-Wolf-based Proof of Retrievability (SW-POR) scheme is proposed to enable the client to check whether the distributed data stored in the cloud servers is intact or not. The proposed SW-POR scheme not only can obtain an optimal coded block size, but also it can provide the exact-repair feature and low complexity. In this paper, the security analysis and efficiency analysis are provided. Simulation results show that the SW-POR scheme can accomplish a significant improvement in computation time.**

***Keywords*-Proof of Retrievability, Network Coding, Slepian-Wolf Coding, Cloud Storage**

I. Introduction

The clients that can be individuals or organizations outsource their data to the cloud storage providers, which allow the clients to access, manage, and share their data portably and easily from anywhere via the Internet. However, the cloud storage providers are untrustworthy, and they thus introduce numerous security challenges: data availability, data integrity, and data confidentiality. This paper focuses on ensuring data availability and data integrity. These challenges are more important than data confidentiality because these challenges are the pre-conditions for the existence of cloud system. To assure these challenges, three common techniques can be used: *replication*, *erasure coding* and *network coding*. Replication, which allows the client to store file replicas in servers, is firstly proposed in [1]. When a corrupted server is detected, the client uses the healthy replicas to repair it. However, the drawback of this technique is high storage cost because the client must store a whole file in each server. Erasure coding [2] is then applied to reduce the storage cost. Erasure coding allows the client to store file blocks in each server redundantly instead of file replica as replication. However, when the corrupted data is repaired, the client has to retrieve the entire original file before the client generates new coded blocks. Therefore, its computation and communication costs are increased during data repair. To improve the efficiency in the data repair, network coding is applied [3]–[5]. The main advantage of network coding is that the client does not need to retrieve the entire file before the client generates new coded blocks.

To restore assurances eroded by cloud storage, many researchers use a protocol called Proof of Retrievability (POR) [6]–[8]. Based on the POR, many schemes have been proposed, for instance [9], [10] using replication and [11]–[13] using erasure coding. Besides that, a few notable network coding-based POR schemes can be found. First, Dimakis et al. [14] apply the network coding to distributed storage systems and achieve a remarkable reduction in the communication overhead of the repair component. Li et al. [15] propose a 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 algorithm. Then, Chen et al. [16] introduce Remote Data Checking for Network Coding (RDC-NC), which provides an elegant solution for efficient data repair by recoding encoded blocks in the healthy servers. H. Chen et al. [17] propose a scheme called NC-Cloud to improve the cost-effectiveness of repair using the functional minimum-storage regenerating (FMSR) codes and lighten the encoding requirement of storage nodes during repair. However, most of these network coding-based POR schemes have not addressed the following shortcomings: (i) the new coded block is not in exactly the same form as the corrupted coded block, (ii) the size of a coded block are greater than or equal to the size of each file block, and (iii) although most of prior papers focus on efficiency, their models still incur high storage, computation, and communication costs.

Contribution. This paper proposes a new POR scheme named Slepian-Wolf-based POR (SW-POR) to address the aforementioned shortcomings. Our contributions are:

- The SW-POR scheme is proposed to ensure the size of a coded block is small compared with the network coding POR schemes.
- In the SW-POR, the new coded block is exactly repaired to be the same as the corrupted coded block unlike other network coding POR schemes. In other words, the proposed SW-POR scheme can provide an exact-repair feature. Furthermore, the additional overheads are unnecessary when the exact-repair is executed. This exact-repair feature also permits the SW-POR's code to be systematic.

- The storage, communication and computation costs in the SW-POR scheme are lower than the costs in other network coding POR schemes. In a real cloud system, the size of the original file is huge, SW-POR definitely becomes beneficial.

The arrangement of this paper is organized as follows. The architecture of cloud storage service, Proof of Retrievability, the background of network coding and Slepian-Wolf coding, and the notations and definitions are described in Section II. The proposed SW-POR scheme is described in Section III. The analysis of security and efficiency is discussed in Section IV and Section V, respectively. The numerical simulation and results are presented in Section VI. Finally, the conclusion is drawn in Section VII.

II. Research Background

A. Architecture

An architecture of cloud storage service is depicted in Figure 1. A client is an entity that has data to be stored in the cloud and relies on the cloud for data storage, computation, and maintenance. This client can be either enterprise or individual customers. Meanwhile, a cloud server is an entity, which is managed and monitored by a cloud service provider to accommodate a service of data storage and has significant and unlimited storage space and computation resources. In the cloud storage service, a client can store his/her data into a set of servers in a simultaneous and distributed manner. Throughout this paper, the terms of 'source' and 'receiver' are indistinguishably used to represent the terms of 'client' and 'server', respectively.

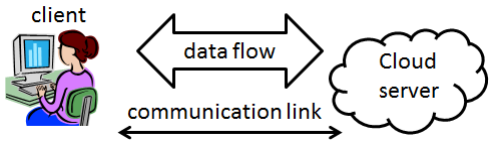

Figure 1: Architecture of cloud storage service

B. Proof of Retrievability

Proof of Retrievability (POR) [6]–[8] is a protocol in which a server proves to a client that a stored data can be fully recovered. POR sometimes is also called as a challenge-response protocol between a client and a server. A POR has a tuple function of (keygen, encode, check, retrieve, repair) as follows:

- keygen(1^λ): It contains a probabilistic algorithm, which is performed by the client with a given security parameter (λ) to produce a secret key (sk) and a public key (pk). For a symmetric key system, pk is set to be null.
- encode(sk, F): This function allows the client to encode a raw file (F) to an encoded file (F'), then stores F' into the server.
- check(sk): It is a function that is conducting the challenge-response protocol in between the client and the server during which the client uses sk to generate a challenge (c) and sends the c to the server. The server computes a corresponding response (r) and sends the r back to the client. The client then verifies the server based on c and r.
- retrieve$(\mathbf{c}_0, \cdots, \mathbf{c}_{m-1})$: This function is performed when the client wants to retrieve the original F based on a set of coded blocks of any servers. This function selects m servers and requests those selected servers to send their coded blocks ($\mathbf{c}$).
- repair(): When a corrupted data from a server is detected in the check function, this function is executed by the client to repair the corrupted data. The repair function is depended on the used techniques, e.g., replication, erasure coding, or network coding.

This paper focuses on the functions of encode, retrieve and repair. The check function is beyond the scope of this paper. This is because several existing schemes are available for the check function. In those schemes, two popular methods are usually used: homomorphic MAC (Message Authentication Code) [18]–[20] and homomorphic signature [21]–[23]. Because this paper does not deal with the check function, the keys are not needed for the checking. Thus, the keygen is also not described in this paper.

C. Network Coding and its Application.

Network coding (NC) [3]–[5] is proposed to improve the network throughput and the efficiency of data transmission and data repair. Suppose that a source owns an file F and wants to send F to a receiver. The source firstly partitions the file into m blocks $\bar{\mathbf{w}}_1, \cdots, \bar{\mathbf{w}}_m$. 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 single '1' in the i-th position and '0' elsewhere. Let $\mathbf{w}_1, \cdots, \mathbf{w}_m$ be the augmented blocks. Each $\mathbf{w}_i$ has the form:

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

The source then sends $\{\mathbf{w}_1, \cdots, \mathbf{w}_m\}$ as packets to the network. When an intermediate node receives k packets $\mathbf{w}_{i_1}, \cdots, \mathbf{w}_{i_k}$, the intermediate 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 packet $\mathbf{w}$ carries m accumulated coding coefficients in $\mathbb{F}_q$ that produce $\mathbf{w}$ as a linear combination of all m original file blocks. The receiver receives the combinations of the original blocks and it can retrieve the original file from any set of m combinations. If $y \in \mathbb{F}_q^{l+m}$ is a linear combination of $\mathbf{w}_1, \cdots, \mathbf{w}_m \in \mathbb{F}_q^{l+m}$, then the linear combination coefficients are contained in the last m coordinates of y.

Application in Distributed Systems. A few researchers have been applied the NC technique in the distributed systems [14]–[17]. From m augmented blocks $\mathbf{w}_1, \cdots, \mathbf{w}_m$, the client chooses m coding coefficients $\alpha_1, \cdots, \alpha_m$ and computes coded blocks as $c = \sum_{i=1}^{m} \alpha_i \cdot \mathbf{w}_i$, then stores those coded blocks in the servers. When a corruption is detected, the client retrieves coded blocks from healthy servers and linearly combines them to regenerate new coded blocks. An example of this mechanism is given in Figure 2.

In Figure 2, the new coded blocks ($5\mathbf{w}_1+5\mathbf{w}_2+16\mathbf{w}_3$ and $8\mathbf{w}_1+8\mathbf{w}_2+27\mathbf{w}_3$) are not the same as the corrupted coded blocks ($2\mathbf{w}_1+2\mathbf{w}_3$ and $\mathbf{w}_2+2\mathbf{w}_3$). Furthermore, since each file block is augmented with a vector of length m, the size of each augmented block $\mathbf{w}_i$ is: $|\mathbf{w}| = m+|\bar{\mathbf{w}}| = m+\frac{|F|}{m}$. The size of a coded block $\mathbf{c}_i$ is equal to the size of a augmented block because the linear combination works in $\mathbb{F}_q$: $|\mathbf{c}| = |\mathbf{w}|$. Therefore, $|\mathbf{c}| = m+\frac{|F|}{m}$, which will be used in the Section V Efficiency Analysis later.

The above-mentioned NC concept is the linear NC in which the coefficients $\alpha_1, \cdots, \alpha_m$ are chosen in $\mathbb{F}_q$ and a coded block is computed using the sum operation. In the case of the XOR-based NC, these coefficients are chosen in $\{0, 1\}$ and a coded block is computed using the XOR operation, but its mechanism is still the same as the linear NC. In this paper, the linear NC is considered for the simulation.

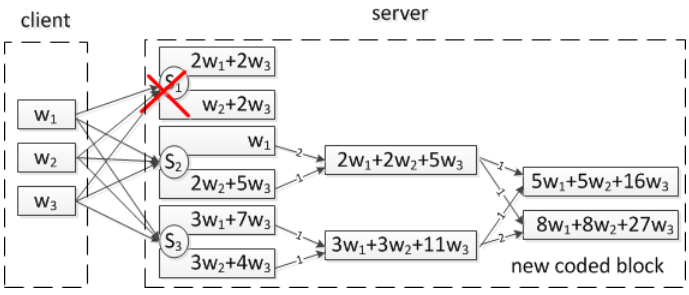

Figure 2: *From augmented blocks $\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}$, the client computes six coded blocks and stores two coded blocks in the server $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$. Suppose $\mathcal{S}_1$ is corrupted, the client requires $\mathcal{S}_2$ and $\mathcal{S}_3$ to create new blocks using linear combination, and then mixes these to obtain two new coded blocks.*

D. Slepian-Wolf Coding

This paper uses the *binning idea* of Slepian-Wolf coding (SWC) [24]. Assume that the source has two blocks X and Y which have the same size in the unit of bits. 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 that $|X|$ is partitioned into t bins each with $\frac{|X|}{t}$ elements. Without SWC, $\log_2|X|$ bits are needed to convey the input to the decoder. With SWC, only $\log_2 t$ bits are needed 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.

Table I: List of notations and definitions.

$\mathcal{C}$	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	constructed block based on XOR for $\mathbf{c}_i$
$\hat{\mathbf{c}}_i$	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	third-operand index of v_i
$\|x\|$	size of x in the unit of bits
$\|\bar{\mathbf{w}}\|$	size of $\bar{\mathbf{w}}_i$, $\|\bar{\mathbf{w}}\| = \frac{\|F\|}{m}$
$\oplus$	XOR operator
$H(x)$	Shannons entropy of a random variable x

E. Notations and Definitions

Throughout this paper, the list of notations and definitions is given in Table I.

III. Proposed SW-POR Scheme

To mitigate the limitations of POR scheme, in this paper a Slepian-Wolf-based POR (SW-POR) is proposed. The form of a coded block $\mathbf{c}_i$ is different from that of the NC. The key idea is that each coded block is the *bin index* as in SWC to achieve better coded block size.

A. Encode Function

From m file blocks, the number of XORs combined from any three different file blocks is $\binom{m}{3}$. However, only n out of $\binom{m}{3}$ XORs are needed. The idea for choosing such n XORs is that:

Remark 1: $\mathcal{C}$ chooses $\bar{\mathbf{w}}_0$ as the first operand, $\bar{\mathbf{w}}_1$ as the second operand, $\bar{\mathbf{w}}_2, \cdots, \bar{\mathbf{w}}_{m-1}$ 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 $\mathcal{C}$ has 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 the NC in which the coded blocks are distributed randomly to the servers. The coded block in $\mathcal{S}_0, \cdots, \mathcal{S}_{n-1}$ 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.

The Encode algorithm is described in Algorithm 1. Given the inputs m, n, $\{\bar{\mathbf{w}}_0, \cdots, \bar{\mathbf{w}}_{m-1}\}$ and $|F|$, the algorithm

Algorithm 1 Encode algorithm

Input: $m, n, \{\bar{\mathbf{w}}_0, \cdots, \bar{\mathbf{w}}_{m-1}\}, |F|$.
Output: $\{\mathbf{c}_0, \cdots, \mathbf{c}_{n-1}\}$, $\{\hat{\mathbf{c}}_0, \cdots, \hat{\mathbf{c}}_{n-1}\}$.

1: $count \leftarrow 0$
2: **for** $\alpha \leftarrow 0$ **to** $m-3$ **do**
3: **for** $\beta \leftarrow \alpha+1$ **to** $m-2$ **do**
4: **for** $\gamma \leftarrow \beta+1$ **to** $m-1$ **do**
5: $\hat{\mathbf{c}}_i \leftarrow 0$
6: **for** $x \leftarrow 0$ **to** $|\bar{\mathbf{w}}_\alpha|-1$ **do**
7: **if** $(\bar{\mathbf{w}}_\alpha[x] \oplus \bar{\mathbf{w}}_\beta[x] \oplus \bar{\mathbf{w}}_\gamma[x] == 1)$ **then**
8: $\hat{\mathbf{c}}_i++$
9: **end if**
10: **end for**
11: $P_i \leftarrow list\_all\_combinations(\frac{|F|}{m}, \hat{\mathbf{c}}_i)$
12: $v_i \leftarrow \bar{\mathbf{w}}_\alpha \oplus \bar{\mathbf{w}}_\beta \oplus \bar{\mathbf{w}}_\gamma$
13: **for** $x \leftarrow 0$ **to** $P_i.length-1$ **do**
14: **if** $(P_i[x] == v_i)$ **then**
15: $\mathbf{c}_i \leftarrow x$
16: **break**
17: **end if**
18: **end for**
19: $count++$
20: **if** $(count == n-1)$ **then**
21: **break**
22: **end if**
23: **end for**
24: **end for**
25: **end for**
26: **return** $\{\mathbf{c}_0, \cdots, \mathbf{c}_{n-1}\}, \{\hat{\mathbf{c}}_0, \cdots, \hat{\mathbf{c}}_{n-1}\}$

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. The 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 programming libraries) 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). The 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 XOR. 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 $\bar{\mathbf{w}}_4 = 1010$ ($|\bar{\mathbf{w}}| = 4$). Suppose $n = 8$, 8 coded blocks $\{\mathbf{c}_0, \cdots, \mathbf{c}_7\}$ need to be constructed. To make $\mathbf{c}_0$, $v_0 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 = 0011$ is used (Remark 1). Since the number of bit '1' in v_0 is 2, $\hat{\mathbf{c}}_0 = 2$. Since $\hat{\mathbf{c}}_0 = 2$ and $|\bar{\mathbf{w}}| = 4$, $P_0 = \{0011, 0101, 0110, 1001, 1010, 1100\}$ in which the elements are sorted in an ascending order and indexed as $\{0, \cdots, 5\}$. The important thing is that: since $|P_0| = \binom{4}{2} = 6$, only $\log_2 6 \simeq 3$ bits are needed 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, $\mathbf{c}_0 = 0_{(decimal)} = 000$. Since $|\hat{\mathbf{c}}| \leq |\bar{\mathbf{w}}|$, the number of bits when transform $\hat{\mathbf{c}}_i$ to the binary form is $\leq \log_2 |\bar{\mathbf{w}}| + 1$. The equation holds when $|\bar{\mathbf{w}}|$ is a power of two. Here, $\hat{\mathbf{c}}_0 = 2_{(decimal)} = 10$. $\{\mathbf{c}_0, \hat{\mathbf{c}}_0\}$ are finally sent to the server $\mathcal{S}_0$. In the same way, $\mathbf{c}_1, \cdots, \mathbf{c}_7$ are computed. The results are as follows:

- $v_0 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 = 0011$, $\hat{\mathbf{c}}_0 = 2_{(decimal)} = 10$, $P_0 = \{0011, 0101, 0110, 1001, 1010, 1100\}$, $\mathbf{c}_0 = 0_{(decimal)} = 000$.
- $v_1 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_3 = 0100$, $\hat{\mathbf{c}}_1 = 1_{(decimal)} = 1$, $P_1 = \{0001, 0010, 0100, 1000\}$, $\mathbf{c}_1 = 2_{(decimal)} = 10$.
- $v_2 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_4 = 1111$, $\hat{\mathbf{c}}_2 = 4_{(decimal)} = 100$, $P_2 = \{1111\}$, $\mathbf{c}_2 = 0_{dec} = 0$.
- $v_3 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_3 = 1011$, $\hat{\mathbf{c}}_3 = 3_{(decimal)} = 11$, $P_3 = \{0111, 1011, 1101, 1110\}$, $\mathbf{c}_3 = 1_{dec} = 01$.
- $v_4 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_4 = 0000$, $\hat{\mathbf{c}}_4 = 0_{(decimal)} = 0$, $P_4 = \{\}$, $\mathbf{c}_4 = 0_{dec} = 0$.
- $v_5 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_3 \oplus \bar{\mathbf{w}}_4 = 0111$, $\hat{\mathbf{c}}_5 = 3_{(decimal)} = 11$, $P_5 = \{0111, 1011, 1101, 1110\}$, $\mathbf{c}_5 = 0_{dec} = 00$.
- $v_6 = \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_3 = 1110$, $\hat{\mathbf{c}}_6 = 3_{(decimal)} = 11$, $P_6 = \{0111, 1011, 1101, 1110\}$, $\mathbf{c}_6 = 3_{dec} = 11$.
- $v_7 = \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 \oplus \bar{\mathbf{w}}_4 = 0101$, $\hat{\mathbf{c}}_7 = 2_{(decimal)} = 10$, $P_7 = \{0011, 0101, 0110, 1001, 1010, 1100\}$, $\mathbf{c}_7 = 1_{dec} = 001$.

Algorithm 2 Retrieve algorithm

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

Algorithm 3 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** $P_i.length - 1$ **do**
  if $(count ==_i)$ **then**
    return $P_i[x]$
  end if
end for
return $P_i[count]$

Algorithm 4 GET algorithm

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

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

Algorithm 5 COEF algorithm

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

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

B. Retrieve Function

To retrieve F when a server is corrupted, the constraints of n and m are chosen as $m < n \leq \binom{m}{3}$. n can be reduced if the scheme is extended so that each $\mathcal{S}_i$ stores multiple coded blocks. $\mathcal{C}$ requires m servers to provide their coded blocks, says $\mathbf{c}_{k_0}, \cdots, \mathbf{c}_{k_{m-1}}$). The coded blocks are chosen such that the binary matrix consisting of coefficient vectors of the XORs have full rank.

The Retrieve algorithm is described in Algorithm 2. Given the inputs 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 to construct $\mathbf{c}_{k_i}$ (line 3: SUB denotes the function 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 the 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 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 retrieved as $\bar{\mathbf{w}}_0 || \cdots || \bar{\mathbf{w}}_{m-1}$ (line 9).

Example 2. This example retrieves 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 retrieve F since the matrix consisting of coefficient vectors of v_0, v_1, v_2, v_3 and v_6 has full rank. Since $\mathbf{c}_0 = 000, \hat{\mathbf{c}}_0 = 2$, $P_0 = \{0011, 0101, 0110, 1001, 1010, 1100\}$, map $\mathbf{c}_0$ to obtain $v_0 = 0011$. In the same way, $v_1 = 0100$, $v_2 = 1111$, $v_3 = 1011$ 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 $v_0 = 0011$, u_0 can be obtained as $u_0 = [1, 1, 1, 0, 0, 0011]$. In the same way, $u_1 = [1, 1, 0, 1, 0, 0100]$, $u_2 = [1, 1, 0, 0, 1, 1111]$, $u_3 = [1, 0, 1, 1, 0, 1011]$ and $u_6 = [0, 1, 1, 1, 0, 1110]$. From u_0, u_1, u_2, u_3, u_6, a matrix $\bar{U}$ is constructed as:

$$\bar{U} = \begin{pmatrix} 1,1,1,0,0,0011 \\ 1,1,0,1,0,0100 \\ 1,1,0,0,1,1111 \\ 1,0,1,1,0,1011 \\ 0,1,1,1,0,1110 \end{pmatrix} \overset{Gauss}{\rightarrow} U = \begin{pmatrix} 1,0,0,0,0,1100 \\ 0,1,0,0,0,1001 \\ 0,0,1,0,0,0110 \\ 0,0,0,1,0,0001 \\ 0,0,0,0,1,1010 \end{pmatrix}$$

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

Algorithm 6 Repair algorithm

Input: $\mathcal{S}_y$
Output: $\mathbf{c}_y, \hat{\mathbf{c}}_y$

1: $\alpha_y, \beta_y, \gamma_y \leftarrow GET(m, y)$
2: Choose $a, b \in \{0, m-1\}$ s.t. $a, b \neq \alpha_y, \beta_y, \gamma_y$
3: $\{\alpha_{r_1}, \beta_{r_1}, \gamma_{r_1}\} \leftarrow ascending\_sort(\alpha_y, \beta_y, a)$
4: $\{\alpha_{r_2}, \beta_{r_2}, \gamma_{r_2}\} \leftarrow ascending\_sort(\alpha_y, \gamma_y, b)$
5: $\{\alpha_{r_3}, \beta_{r_3}, z\gamma_{r_3}\} \leftarrow ascending\_sort(\alpha_y, a, b)$
6: $\mathcal{S}_{r_1} \leftarrow REGET(\alpha_{r_1}, \beta_{r_1}, \gamma_{r_1})$
7: $\mathcal{S}_{r_2} \leftarrow REGET(\alpha_{r_2}, \beta_{r_2}, \gamma_{r_2})$
8: $\mathcal{S}_{r_3} \leftarrow REGET(\alpha_{r_3}, \beta_{r_3}, \gamma_{r_3})$
9: 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: $v_{r_1} \leftarrow SUB(\mathbf{c}_{r_1}, \hat{\mathbf{c}}_{r_1}, \frac{|F|}{m})$
11: $v_{r_2} \leftarrow SUB(\mathbf{c}_{r_2}, \hat{\mathbf{c}}_{r_2}, \frac{|F|}{m})$
12: $v_{r_3} \leftarrow SUB(\mathbf{c}_{r_3}, \hat{\mathbf{c}}_{r_3}, \frac{|F|}{m})$
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: $P_y \leftarrow list\_all\_combinations(\frac{|F|}{m}, \hat{\mathbf{c}}_y)$
16: $\mathbf{c}_y \leftarrow ord(P_y, v_y)$
17: **return** $\mathbf{c}_y, \hat{\mathbf{c}}_y$

Algorithm 7 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** $m-3$ **do**
3: **for** $\beta \leftarrow \alpha+1$ **to** $m-2$ **do**
4: **for** $\gamma \leftarrow \beta+1$ **to** $m-1$ **do**
5: $count++$
6: **if** $(\alpha == \alpha_{r_i})$ **and** $(\beta == \beta_{r_i})$ **and** $(\gamma == \gamma_{r_i})$ **then**
7: **return** $count$
8: **end if**
9: **end for**
10: **end for**
11: **end for**

C. Repair Function

The Repair algorithm is described in Algorithm 6. GET, $REGET$ and SUB are the sub-algorithms. Assume that $\mathcal{S}_y$ is 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 XOR $\bar{\mathbf{w}}_{\alpha_y} \oplus \bar{\mathbf{w}}_{\beta_y} \oplus \bar{\mathbf{w}}_{\gamma_y}$ can be replaced as:

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

Thirdly, $\{\alpha_y, \beta_y, a\}$, $\{\alpha_y, \gamma_y, b\}$ and $\{\alpha_y, a, b\}$ are increasingly sorted using the $ascending\_sort$ algorithm (line 3-5) and returned to three lists $\{\alpha_{r_1}, \beta_{r_1}, \gamma_{r_1}\}$, $\{\alpha_{r_2}, \beta_{r_2}, \gamma_{r_2}\}$ and $\{\alpha_{r_3}, \beta_{r_3}, \gamma_{r_3}\}$, respectively. The explanation of $ascending\_sort$ is skipped since it is a simple programming function. Fourthly, $\mathcal{C}$ finds three servers storing three XORs using the $REGET$ algorithm (line 6-8). $\mathcal{C}$ finds the real result of XORs $(v_{r_1}, v_{r_2}, v_{r_3})$ by using the SUB algorithm (line 10-12). 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, the algorithm finds the coded block of $\mathcal{S}_y$ (line 15-16) like the encode function.

Example 3. This example follows 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 increasingly sorted 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 $\{\alpha_{r_1}, \beta_{r_1}, \gamma_{r_1}\} = \{0, 1, 2\}$, the algorithm finds $\mathcal{S}_0$ because $\mathcal{S}_0$ uses $\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2$. Given $\{\alpha_{r_2}, \beta_{r_2}, \gamma_{r_2}\} = \{0, 3, 4\}$, the algorithm finds $\mathcal{S}_5$ because $\mathcal{S}_5$ uses $\bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_3 \oplus \bar{\mathbf{w}}_4$. Given $\{\alpha_{r_3}, \beta_{r_3}, \gamma_{r_3}\} = \{0, 1, 4\}$, the algorithm finds $\mathcal{S}_2$ 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 $\{\mathbf{c}_0 = 000, \hat{\mathbf{c}}_0 = 2\}$, the algorithm finds $v_0 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_2 = 0011$. Given $\{\mathbf{c}_5 = 00, \hat{\mathbf{c}}_5 = 3\}$, the algorithms finds $v_5 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_3 \oplus \bar{\mathbf{w}}_4 = 0111$. Given $\{\mathbf{c}_2 = 0, \hat{\mathbf{c}}_2 = 4\}$, the algorithm finds $v_2 = \bar{\mathbf{w}}_0 \oplus \bar{\mathbf{w}}_1 \oplus \bar{\mathbf{w}}_4 = 1111$ (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 $|\bar{\mathbf{w}}| = 4$, $P_y = \{0111, 1011, 1101, 1110\}$. Since $|P_y| = 4$, $\log_2 4 = 2$ bits are needed to present $\mathbf{c}_y$. The coded block $\mathbf{c}_y$ is the index of v_y in P_y which is: $\mathbf{c}_y = 1_{decimal} = 01$.

From the Example 3, it can conclude that the new coded block in the SW-POR scheme is exactly same as the corrupted coded block unlike the NC scheme.

IV. Security Analysis

A. Constraint for Retrieve Function

In each time step, the servers are checked. If a corrupted server is detected, it is repaired in the next time step.

Theorem 1. *The raw file F can be retrieved as long as in any time step, 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

Table II: A comparison of NC and SW-POR schemes

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

computed using XOR $v_i = \bar{\mathbf{w}}_\alpha \oplus \bar{\mathbf{w}}_\beta \oplus \bar{\mathbf{w}}_\gamma$. To retrieve F, a set of $\bar{\mathbf{w}}_0, \cdots, \bar{\mathbf{w}}_{m-1}$ is viewed as the unknown variables. To solve these m unknown variables, at least m coded blocks are needed to make the matrix have full rank.

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

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

B. Security Threshold

This subsection shows 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 obtain. 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 the XOR operation.

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

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

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 $\hat{\mathbf{c}}_{i_0}, \cdots \hat{\mathbf{c}}_{i_{\lambda-1}}$. The property of conditional entropy leads to:

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

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, thus: $H(F|v_{i_0}, \cdots, v_{i_{\lambda-1}}) = H(F)$. Therefore:

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

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

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

When $m \le \lambda$, $\frac{m-\lambda}{\lambda} H(F) = 0$. Therefore:

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

■

V. Efficiency Analysis

A. Storage Cost

In the NC scheme, 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 stores: $\mathbf{c}_i$ which has the size $|\mathbf{c}| = \log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i}$ and $\hat{\mathbf{c}}_i$ which has the size $|\hat{\mathbf{c}}| = \log_2|\bar{\mathbf{w}}|$ because $\hat{\mathbf{c}}_i \in \{1, |\bar{\mathbf{w}}|\}$. The storage cost in the SW-POR scheme is thus: $O(\log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i} + \log_2|\bar{\mathbf{w}}|)$. $\forall|\bar{\mathbf{w}}|$ 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 $|\bar{\mathbf{w}}| < 2^m$.

B. Computation Cost

1) Encode: In the NC scheme, 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 the NC scheme, $\mathcal{C}$ requires m healthy coded blocks that takes $O(m)$ operations. Similarly, in SW-POR, $\mathcal{C}$ also needs $O(m)$ operations.

3) Repair: In the NC scheme, $\mathcal{C}$ needs at least m coded blocks from m healthy servers for repairing. This has $O(m)$ operations. In SW-POR, only three healthy coded blocks are needed for repairing. This has $O(1)$ operations.

C. Communication Cost

1) Encode: In the NC scheme, 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 the NC scheme, the cost for m servers to send their packets to $\mathcal{C}$ is $O(m(m+|\bar{\mathbf{w}}|))$. 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 the NC scheme, 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, only three healthy servers are needed for data repair. The cost in the SW-POR scheme is thus $O(3(\log_2\binom{|\bar{\mathbf{w}}|}{\hat{\mathbf{c}}_i}+\log_2|\bar{\mathbf{w}}|))$.

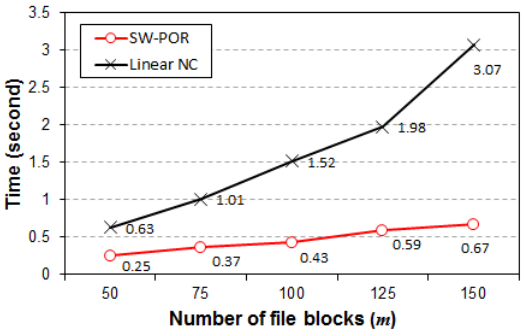

Figure 3: The computation time performance of the Encode function

VI. Numerical Evaluation

This section evaluates the computation performance of the NC scheme and the proposed SW-POR scheme. A program that is 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. For NC scheme, the $|q|$ is set to 2^{10} bits. For both NC and SW-POR schemes, each block, $|\mathsf{w}|$ is set to 2^{10} bits and each server is assumed to store only one coded block. Other parameters such as the number of servers, n is set to $m+1$. We ran our simulations

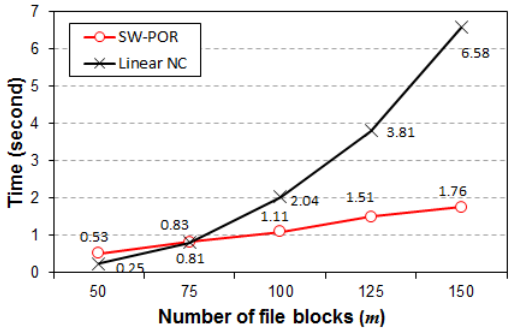

Figure 4: The computation time performance of the Retrieve function

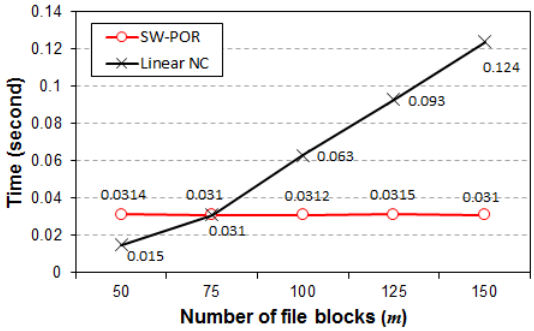

Figure 5: The computation time performance of the Repair function

for 100 simulation sets. Then, our simulation results are averaged. In the simulation, gmpy2 and itertools libraries are used. The simulation results are observed with three sets of computation performance: encode, retrieve, and repair by varying the number of file blocks, m. The simulation results reveal that the computation time increases as the number of file blocks increases for both NC and SW-POR schemes. However, three graphs show that the slope of increment for SW-POR scheme increases merely compared with the NC scheme. It can also conclude that the SW-POR scheme can reduce the computation time by about 78.2% for encode function, by about 73.3% for retrieve function, and by about 75.0% for repair function when the number of file blocks is 150 compared with the NC scheme.

VII. Conclusion

In this paper, the SW-POR scheme has been proposed to obtain an optimal coded block size and to provide the exact-repair feature. The primary key idea is based on the binning idea of SWC, which is a common data compression technique in the communication network. The security threshold of SW-POR is proved based on the entropy theory. Furthermore, the efficiency of SW-POR is analyzed based on the complexity theory. Simulation results reveal that the SW-POR scheme can attain an huge improvement of

computation time for cncode, retrieve, and repair functions by more than 70% when the number of file blocks is 150. Further research is required to investigate the implementation of SW-POR scheme in the real cloud storage system.

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

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

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

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

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

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

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

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

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

[10] 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.

[11] 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.

[12] 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*, Mar 2009, pp.109-127

[13] 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.

[14] 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.

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

[16] 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.

[17] H.C.H. Chen, Yuchong Hu, P.P.C. Lee and Yang Tang, "NC-Cloud: 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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*, pp.244-262, 2002.

[22] 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.

[23] 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.

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