Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 586720, 14 pages
http://dx.doi.org/10.1155/2015/586720

Research Article

MD-POR: Multisource and Direct Repair for Network Coding-Based Proof of Retrievability

Kazumasa Omote and Tran Phuong Thao

Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

Correspondence should be addressed to Tran Phuong Thao; tpthao@jaist.ac.jp

Received 22 October 2014; Accepted 8 January 2015

Academic Editor: Sangjun Lee

When data owners publish their data to a cloud storage, data integrity and availability become typical problems because the cloud servers are never trusted. To address these problems, researchers proposed the Proof of Retrievability (POR) protocol which allows a verifier to check and repair the data stored in the cloud servers. Based on the POR protocol, the network coding technique is commonly applied to increase the efficiency in data transmission and data repair. However, most previous schemes neither consider a practical scenario nor use the network coding efficiently. In this paper, a lightweight network coding-based POR scheme, called MD-POR (Multisource and Direct Repair for Proof of Retrievability) is proposed. Unlike previous schemes, the proposed MD-POR scheme allows multiple clients who have different secret keys to participate in the scheme. Moreover, the MD-POR scheme supports the direct repair feature in which a corrupted data can be recovered by the servers without burdening the clients. The MD-POR scheme also supports public authentication feature in which a third party auditor is employed to check the servers, and the client is thus free of the responsibility of periodically checking the servers. Furthermore, the MD-POR scheme is constructed based on a symmetric key setting.

1. Introduction

Since data is increasing exponentially, database owners trend to publish their data to storage providers called clouds in order to reduce the burden of data storage and maintenance. Clients can thus access, manage, and share their data from anywhere via the Internet. However, such service providers are untrustworthy and present three basic challenges to data security: (i) integrity, (ii) availability, and (iii) confidentiality. In confidentiality, there are two research approaches: the cryptographic approach (e.g., RSA) and the information-theoretic approach (e.g., secret sharing scheme). Compared to the cryptographic confidentiality approach, the information-theoretic confidentiality approach achieves a security level determined by a threshold. We choose the information-theoretic approach because our security analysis derives purely from information theory. In this paper, we deal with integrity, availability and information-theoretic confidentiality.

To check the cloud servers, researchers proposed the Proof of Retrievability (POR) protocol [1–3] that enables the servers (provers) to demonstrate to the verifier whether the data stored in the servers is intact and available and enables the clients to recover the data when an error is detected. Based on the POR protocol, the integrity and availability assurance are mainly based on three techniques: replication [4], erasure coding [5], and network coding [6–9]. In the replication technique, the client stores file replicas in each server. When a corrupted server is detected, the client uses one of 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 technique 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. Network coding technique is then applied to improve the efficiency in the data repair. 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.

Consequently, in this paper, we focus on the network coding technique. Our goal is to construct a network-coding POR which satisfies the following aims.

(i) Practical scenario: the system should consist of multiple clients, each client keeps a different secret key. This is because in many distributed storage systems today such as Dropbox, each client has a personal data; and hence, each client should use his own secret key to satisfy integrity and confidentiality.

(ii) Lightweight: firstly, the clients should be free of two heaviest tasks: periodically checking the servers and repairing the corrupted servers. Secondly, the system should be constructed in a symmetric key setting which is a well-known lightweight cryptography rather than an asymmetric key setting.

Network Coding-Based POR Schemes. A few notable network-coding PORs were proposed. Dimakis et al. [10] were the first applying network coding to the distributed storage system. Li et al. [11] proposes a tree-structure data regeneration for the network coding to optimize network bandwidth by using a maximum spanning tree. Chen et al. [12] then adapted the scheme of Dimakis et al. to propose the Remote Data Checking for Network Coding-based distributed storage system (RDC-NC) scheme which provides an elegant data repair by recoding encoded blocks in healthy servers during repair. Cao et al. [13] applied the Luby transform (LT) code for reducing the computation cost because the LT code is a special network code which works in the finite field of order two and only uses exclusive-OR (XOR) operation. Chen et al. [14] proposed the NC-Cloud scheme to improve the cost-effectiveness of repair using the functional minimum storage regenerating (FMSR) code and lighten the encoding requirement of storage nodes during repair. However, all these schemes cannot hold our aims. These system models only have a single client. Furthermore, the check and repair phases in these schemes bring a lot of burden to the client because (i) the client has to periodically check the servers and (ii) when a corrupted server is detected, the healthy servers provide their blocks to the client; the client then has to verifies them, computes the new blocks, and sends these new blocks to the new server. Le and Markopoulou after that proposed the NC-Audit scheme [15] in which a third party auditor is employed and is delegated the responsibility to check the servers instead of the client. The authors also discussed a new repair mechanism in which the new server can compute the new blocks by itself without the need of the client. We call that mechanism as direct repair. Unfortunately, their direct repair is not completed because they mainly focused on how to prevent the data leakage from the third party auditor. Furthermore, their scheme is constructed in an asymmetric key setting and does not deal with multiple clients.

Contribution. In this paper, a new network-coding POR named as MD-POR is proposed. To the best of our knowledge, we are the first to propose a symmetric key setting-based direct repair for the POR; furthermore, the proposed MD-POR scheme also supports multiclient and public authentication.

(i) *Direct Repair*. If a corrupted server is detected, the healthy servers are required to provide their coded blocks directly to the new server instead of sending these coded blocks back to the client. Afterwards, the new server verifies the coded blocks it received and computes the new coded blocks for itself without disturbing the client. This mechanism can reduce the communication cost and the burden for the client.

(ii) *Multiclient*. To enable multiple clients, our method does not simply duplicate the process of a single client to multiple parallel processes for multiple clients. Instead, in the proposed MD-POR scheme, the processes of multiple clients are mixed together without loosing the data confidentiality of individual clients. To enable such a multiclient setting, we employ the InterMac technique [16] which was proposed for network scenario. The InterMac technique allows multiple sources to send their packages to the network using different secret keys and allows the recipients to verify the packages they received.

(iii) *Symmetric Key Setting*. The MD-POR scheme uses only secret keys without any public key, unlike an asymmetric key setting.

(iv) *Public Authentication*. Not only the client but also any entity who has a given information can check the cloud servers while learning nothing about the secret key of each client. We employ a third party auditor (TPA) on behalf of the clients to check the servers periodically. By delegating the responsibility of checking the servers to the TPA, the clients are free of the burden of checking the servers. Otherwise, for the nonexistence of TPA, the clients have to periodically check the servers, and the public authentication feature cannot be supported because only the clients can check the servers. Although the MD-POR scheme supports public authentication, our method does not use an asymmetric key setting.

Organization. The system model, the backgrounds of the Proof of Retrievability, the network coding technique, the InterMac technique, the notations, and definitions are described in Section 2. The adversarial model is presented in Section 3. The MD-POR scheme is proposed in Section 4. The security analysis and efficiency analysis are given in Section 5 and Section 6, respectively. The performance evaluation of the MD-POR scheme is shown in Section 7. The conclusion and future work are drawn in Section 8.

2. Preliminaries

2.1. System Model. The system model of the MD-POR scheme is depicted in Figure 1. There are three types of entities.

(i) Clients: these entities have data to be stored in the cloud and rely on the cloud for data storage, computation, and maintenance. These clients can be either enterprises or individual customers.

(ii) Cloud servers: the cloud servers are managed and monitored by a cloud service provider to accommodate a service of data storage and have significant and unlimited storage space and computation resources. In the cloud storage service, the clients can store their data into a set of servers in a simultaneous and distributed manner.

(iii) Third party auditor (TPA): this entity is delegated the responsibility to check the servers on behalf of the clients. The TPA is assumed to be trusted to perform the task of periodically checking the servers.

Originally, the system model which consists of only the client and the servers without the TPA is enough for data check. To enable the public authentication feature, the TPA is employed with an assumption that the TPA is a honest-but-curious entity. Several previous papers also use the same assumption of the TPA, for example, [15, 17–19].

2.2. Proof of Retrievability (POR). To check the servers, researchers proposed the Proof of Retrievability (POR) [1–3] which is a challenge-response protocol between a verifier (client) and a prover (server). The POR has four phases as follows.

(1) keygen(1^{λ}): given a security parameter λ, the client runs this algorithm to generate a secret key (sk) and a public key (pk). For the symmetric key setting, pk is set to be null.

(2) encode(sk, F): the client runs this algorithm to encode an original file (F) to an encoded file (F') and then sends F' to the server to store.

(3) check(sk): the client uses his secret key sk to generate a challenge (c) and sends c to the server. The server then computes a response (r) and sends r back to the client. Finally, the client verifies whether the file F is intact based on c and r.

(4) repair(): the client runs this algorithm only when a failure is detected in the check phase. The technique of the repair phase depends on each specific technique, for example, replication, erasure coding, or network coding.

To be suitable for our system model, we modify the POR such that the verifier is the TPA and there are multiple clients as follows.

(1) keygen(1^{λ}): given a security parameter λ, the algorithm generates a set of secret keys $\{\text{sk}_i\}_{i\in\{1,\dots,s\}}$ for s clients and a secret key κ for the TPA.

(2) encode(sk_i, F_i): each client i uses his secret key sk_i to encode his original file F_i to an encoded file F_i' and then sends F_i' to the servers. Each server then linearly combines all F_i' ($i \in \{1,\dots,s\}$) and stores the combined blocks.

(3) check(κ): the TPA uses his key κ to generate a challenge c and sends c to the servers. Each server then computes a response r and sends r back to the TPA. Finally, the TPA verifies whether each F_i is intact or not.

(4) repair(): this algorithm is executed when a failure is detected in the check phase. The technique of the repair phase depends on each specific scheme.

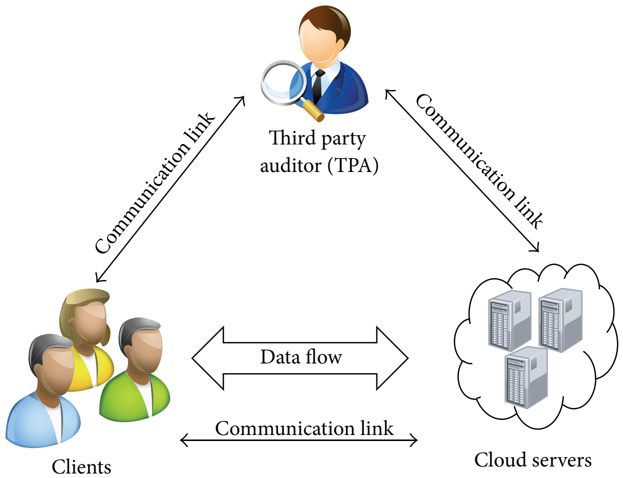


Figure 1: System model.

2.3. Network Coding. Network coding [6–9] is commonly used in network transmission to obtain a good trade-off in term of bandwidth and data repair. Network coding is proposed firstly for the network scenario. It then is applied to the distributed storage system scenario.

Fundamental Concept. In the network scenario, suppose that a source node C wants to send its message to a receiver node R. Before transmitting, C breaks the message into m blocks $v_1,\dots,v_m$; each file block belongs to $\mathbb{F}_q^n$ where $\mathbb{F}_q^n$ denotes a n-dimensional vector space over a finite field $\mathbb{F}$ with a prime q. C augments each file block v_i ($i \in \{1,\dots,m\}$) with a vector of length m in which a single "1" is in the ith position and "0's are elsewhere. Let $w_1,\dots,w_m$ be the augmented blocks. Each augmented block has the following form:

$$w_i = \left(v_i, \overbrace{0,\dots,0,\underbrace{1}_{i},0,\dots,0}^{m}\right) \in \mathbb{F}_q^{n+m}. \qquad (1)$$

These augmented blocks are then sent as packets to the network. When an intermediate node I in the network receives t packets, I will generates t coefficients, linearly combines t packets using the generated coefficients, and transmits the result to its adjacent nodes. Consequently, the receiver node R can receive combinations of all augmented blocks. R can recover m augmented blocks using any set of m combinations. Suppose that R receives m packages $y_1,\dots,y_m \in \mathbb{F}_q^{n+m}$, and R solves all m augmented blocks $w_1,\dots,w_m \in \mathbb{F}_q^{n+m}$ using the accumulated coefficients which are contained in the last m coordinates of each package y. Afterwards, the file blocks $v_1,\dots,v_m$ can be obtained from the first coordinate of each augmented block. Finally, the original message can be reconstructed by concatenating all file blocks.

Application in Distributed Storage System. In the network scenario as described above, there are multiple types of entities: source node, intermediate nodes, and receiver node. However, when the network coding is applied to the distributed storage system scenario, there are two types of entities: a client and servers. Suppose that a client has the original file F which consists of m file blocks $(v_1, \dots, v_m)$. The client wants to store redundantly encoded blocks in the servers in a way that the client can reconstruct the original file F and can repair the encoded blocks in a corrupted server. From these file blocks, the client firstly creates m augmented blocks $(w_1, \dots, w_m)$. The client then chooses m coding coefficients $(\alpha_1, \dots, \alpha_m \in \mathbb{F}_q)$ and computes coded blocks using the linear combination as $c = \sum_{i=1}^{m} \alpha_i \cdot w_i$ and then stores these coded blocks in the servers. To reconstruct the original file F, any m coded blocks are required to solve m augmented blocks $w_1, \dots, w_m$ using the accumulated coefficients contained in the last m coordinates of each coded block. After these m augmented blocks are solved, m file blocks $v_1, \dots, v_m$ are obtained from the first coordinate of each augmented block. Finally, the original file F is reconstructed by concatenating the file blocks. Note that the matrix consisting of the coefficients used to construct any m coded blocks should have full rank. Koetter and Medard [20] proved that if the prime q is chosen large enough and the coefficients are chosen randomly, the probability for the matrix having full rank is high. Once a corrupted server is detected, the client repairs it as follows: the client retrieves coded blocks from the healthy servers and linearly combines them to regenerate new coded blocks. An example about the data repair of network coding is given in Figure 2.

2.4. InterMac. Before describing how the InterMac works, we explain why it is used in our proposed MD-POR scheme as follows. We consider a network in which multiple sources are simultaneously supported and each source owns a different secret key. The data of each source cannot be checked alone. Instead, each source uses the secret key to compute an additional information which is Message Authentication Code (MAC) for each data block. A MAC is also called as *tag*. Each source then transmits the packets consisting of the data blocks and the corresponding tags to the next adjacent node in the network. A node in the network will linearly combine the received blocks and the homomorphic tags. *Herein lies the difficulty of the task: when a recipient node receives a packet, how can this node verify the received linear blocks based on the linear homomorphic tags without any information about any of the secret keys.* The traditional methods, that is, MAC or HMAC, are inadequate to solve this task. Some recent schemes related to this problem have been proposed, for example, [21–23]; unfortunately, they all use an asymmetric key setting, which is not our aim.

The InterMac technique [16] is a suitable technique to generate such secret keys for multiple sources. The characteristic of this technique is that the key of the source $\mathscr{C}_p$ $(p \in \{1, \dots, s\}$ where s denotes the number of sources) is orthogonal to all the augmented blocks which do not belong to $\mathscr{C}_p$. This characteristic can help the verifier check the received packets without needing the information on any of the secret keys.

Construction. Let $w_{11}, \dots, w_{sg} \in \mathbb{F}_q^{n+m}$ be the augmented blocks that have span π, and let them represent as row vectors (where s denotes the number of sources, g denotes the number of file blocks per source, and $m = s \cdot g$). For each $p \in \{1, \dots, s\}$, let M_p be the matrix whose rows are vectors in the following set:

$$\left\{w_{ij} \mid i = 1, \dots, s;\ i \neq p;\ j = 1, \dots, g\right\}. \tag{2}$$

In other words, M_p is the matrix consisting of the augmented blocks of all other sources except $\mathscr{C}_p$. $\operatorname{rank}(M_p) = m - g$. Let π_{M_p} denote the space spanned by the rows of M_p.

The null space of M_p, denoted as $\pi_{M_p}^{\perp}$, is the set of all row vectors $z \in \mathbb{F}_q^{n+m}$ for which $M_p z^T = 0$. For any $(m-g)\times(n+m)$ matrix M_p, we have

$$\operatorname{rank}\left(M_p\right) + \operatorname{nullity}\left(M_p\right) = n + m \tag{3}$$

known as rank-nullity theorem, where $\operatorname{nullity}(M_p)$ is the dimension of $\pi_{M_p}^{\perp}$. Hence,

$$\dim\left(\pi_{M_p}^{\perp}\right) = n + m - (m - g) = n + g. \tag{4}$$

Let $b_1, \dots, b_{n+g} \in \mathbb{F}_q^{n+m}$ be a basis of $\pi_{M_p}^{\perp}$. This basis can be found by solving $M_p z^T = 0$. Let F be a pseudorandom function (PRF): $\overline{\mathscr{K}} \times ([1, s] \times [1, n + g]) \rightarrow \mathbb{F}_q$. A key k_p for the source $\mathscr{C}_p$ is computed as follows:

(i) $r_i \leftarrow F(k, p, i) \in \mathbb{F}_q,\ \forall i \in \{1, \dots, n + g\}$;

(ii) $k_p \leftarrow \sum_{i=1}^{n+g} r_i b_i \in \mathbb{F}_q^{n+m}$.

Eventually, a key set $\{k_1, \dots, k_s\}$ is generated in which each key k_p where $p \in \{1, \dots, s\}$ is constructed as above.

2.5. Notations and Definitions. Throughout this paper, the list of notations and definitions is given in Notation section.

3. Adversarial Model

In the MD-POR scheme, only the clients are trusted because they are the data owners. The following entities are untrusted and considered to be adversaries:

(i) attackers outside the system;

(ii) the cloud servers in the system;

(iii) the TPA in the system (the TPA is assumed not to collude with the servers. We explained about this assumption in Section 2.1).

Concretely, the adversaries can perform the following the attacks.

3.1. Mobile Attack. This attack is performed by an adversary $\mathscr{A}$ outside the system. $\mathscr{A}$ potentially corrupts all the servers across the full system lifetime. A restriction on $\mathscr{A}$ is that he/she can control only $(h - l)$ out of h servers in any given

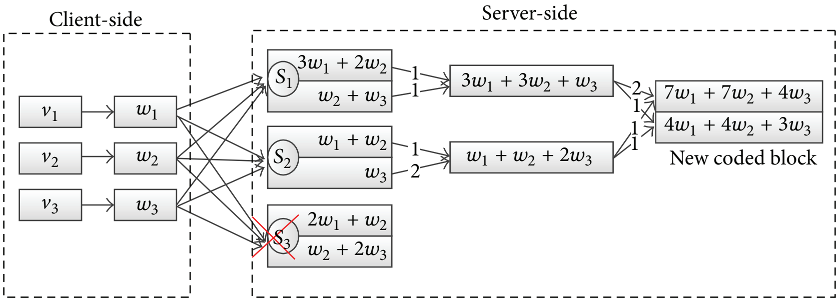


FIGURE 2: From three augmented blocks $\{w_1, w_2, w_3\}$, the client computes six coded blocks and stores two coded blocks in each of servers S_1, S_2, S_3. Suppose that S_3 is corrupted, the client requires S_1 and S_2 to create new blocks using linear combination, and then the client mixes them using linear combination to obtain two new coded blocks and stores them in the new server.

time step. Let *epoch* denote a given time step. In each epoch, the servers are checked. If a corruption is detected on a certain server, the blocks stored in that corrupted server will be repaired from redundancy in the intact servers. Without the server checks, the adversary $\mathscr{A}$ can corrupt all the servers of the system in $h/(h-l)$ epochs.

3.2. Curious Adversary. This attack is performed by the TPA or a new server. In the check phase, the TPA is given a key κ which is constructed from all the secret keys of the clients. In the repair phase, a new server is given another key κ' which is also constructed from all the secret keys of the clients. When they are given their keys, these adversaries try to learn the secret keys because once all secret keys are obtained, these adversaries can fake a valid response when they are checked.

3.3. Response Forgery. This forgery is performed by the servers. In the check phase, the verifier checks all the servers to ensure that they are not corrupted. Each server has to send a response to the verifier in order to demonstrate that the server is healthy. However, a checked server may forge the response to deceive the verifier. If the forged response from the adversarial server satisfies the verification, that server can pass the check phase.

3.4. Pollution Attack. This attack is performed by the servers. The purpose of this attack is to break the linear independence of the encoded blocks. In a network, if a node is malicious and forward invalid package, receivers then obtain multiple packets and cannot tell which of their received packets are corrupt. In other words, the purpose of this attack is to inject invalid packets to prevent data recover. In the POR, this attack happens when a malicious server uses correct data to pass the check phase but then provides invalid data in the repair phase. For example, the client encodes the augmented blocks w_1, w_2, and w_3 to six coded blocks: c_{11}, c_{12} (stored in the server S_1), c_{21}, c_{22} (stored in the server S_2), and c_{31}, c_{32} (stored in the server S_3). In the check phase, suppose that S_3 is detected as being corrupted. Then, in the repair phase, S_3 should be repaired using two coded blocks: c'_{31} (which is a linear combination of c_{11} and c_{12}) and c'_{32} (which is a linear combination of c_{21} and c_{22}). However, at this time, S_1 is malicious without being detected because this time is the repair phase, not the check phase any more. The client still thinks S_1 is healthy; thus, to recover S_3, the client requests coded blocks from S_1 and S_2 but S_1 will provide an invalid coded blocks c''_{31} to the client instead of c'_{31}.

4. The Proposed MD-POR Scheme

Before describing the proposed MD-POR scheme in detail, the technical roadmap is depicted in Figure 3. The file blocks are used to generate the augmented blocks. Then, the augmented blocks are combined with random values to compute the keys. Meanwhile, the augmented blocks are linearly combined into the coded blocks using the network coding. Finally, the coded blocks are tagged using the keys. The coded blocks and the tags are the outputs. The network coding is used because it is related to the repair feature (Section 2.3). The InterMac is used because it is related to the multiuser feature (Section 2.4). Both the network coding and the InterMac are constructed based on linear combinations; therefore, they are suitable to combine together in the proposed scheme.

Let $\mathscr{C}_1, \ldots, \mathscr{C}_s$ be the set of s clients. Each client $\mathscr{C}_i$ ($i \in \{1, \ldots, s\}$) keeps a secret key k_i and has a file $F_i = (v_{i1}, \ldots, v_{ig})$ where g is the number of file blocks. Each file block $v_{ij} \in \mathbb{F}_q^n$ ($j \in \{1, \ldots, g\}$). $\mathscr{C}_i$ creates g augmented blocks $(w_{i1}, \ldots, w_{ig})$ from g file blocks $(v_{i1}, \ldots, v_{ig})$. Each augmented block w_{ij} has the form as in [16]

$$w_{ij} = \left(v_{ij}, \underbrace{\underbrace{0, \ldots, 0}_{g(i-1)}, \underbrace{\overbrace{0, \ldots, 0, 1}^{j}, 0, \ldots, 0}_{g}, \underbrace{0, \ldots, 0}_{g(s-i)}}_{m=sg} \right) \in \mathbb{F}_q^{n+m}, \tag{5}$$

where $i \in \{1, \ldots, s\}$, $j \in \{1, \ldots, g\}$, and $m = sg$.

Each client $\mathscr{C}_i$ uses his secret key k_i to compute the tag t_{ij} for each augmented blocks w_{ij}. The augmented blocks and the tags are then linearly combined and transmitted to all the servers. In every epoch, when the servers are checked by the TPA, the servers have to combine the coded blocks and the tags again and send them back to the TPA. The TPA can finally verify the aggregated coded blocks and the tags

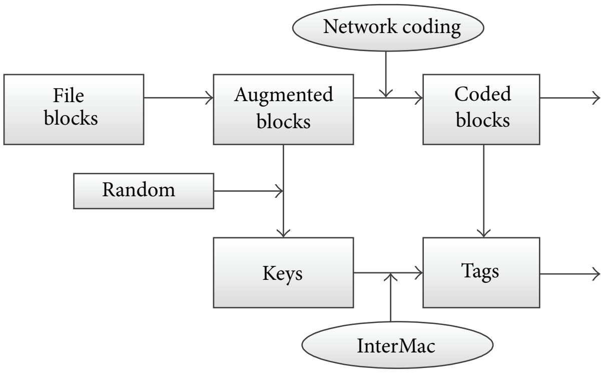


Figure 3: Technical roadmap.

even though the TPA does not know any secret key $\{k_i\}$ $(i \in \{1,\dots,s\})$.

The proposed MD-POR scheme is now described in detail via each phase of the POR as follows.

4.1. Keygen

4.1.1. Keys for the Clients (Keygen1). Each key k_p of the client $\mathscr{C}_p$ $(p \in \{1,\dots,s\})$ is constructed in such a way that k_p is orthogonal to all the augmented blocks which do not belong to $\mathscr{C}_p$. In a formal statement, k_p is constructed as

$$\forall i \in \{1,\dots,s\},\ i \neq p,\ p \in \{1,\dots,s\},\quad w_{ij} \cdot k_p = 0. \tag{6}$$

Using the InterMac (Section 2.4), a key set $\{k_1,\dots,k_s\}$ is created. Then, each $k_p \in \mathbb{F}_q^{n+m}$ is assigned to the client $\mathscr{C}_p$ as the secret key, and the sum of all the keys $\kappa = k_1 + \cdots + k_s \in \mathbb{F}_q^{n+m}$ are assigned to the TPA via a secure channel. The security of the secret keys will be proved later.

4.1.2. Dynamic Keys for a New Server (Keygen2). When a repair phase is executed, the new server will be given a key $\kappa' = (k_1 + \cdots + k_s) + k_{\text{repair}} = \kappa + k_{\text{repair}}$. The new server will use the key κ' to check pollution attack during the repair phase. κ is already computed in Keygen1. Only k_{repair} is different in each repair time. This is to ensure that an adversary cannot attack the new server to obtain k_{repair} for passing the pollution attack check in the later repair phases (we thereafter explain in Section 5.4). When k_{repair} is constructed in the first time, the basis of $b_1,\dots,b_{n+g}$ is computed and saved for the later times. In the next repair times, the basis will be reused to save the computation cost, and only the random coefficients r_i are regenerated again to compute k_p.

r_{repair} has to be orthogonal to all augmented blocks of all the clients. Keygen2 is quite similar to Keygen1. However, the different thing is that $p \notin \{1,\dots,s\}$, p is randomly chosen in $\mathbb{F}_q$ such that $p > s$ in every repair time. Since r_{repair} is orthogonal to all augmented blocks of all the clients, M_p is now the matrix consisting of all the augmented blocks of all the clients. Put differently, the rows of M_p are vectors in the following set:

$$\{w_{ij} \mid i = 1,\dots,s;\ j = 1,\dots,g\}. \tag{7}$$

The set consists of $m = sg$ augmented blocks and each augmented block belongs to $\mathbb{F}_q^{n+m}$. For the $m \times (n+m)$ matrix M_p, the rank-nullity theorem yields

$$\operatorname{rank}(M_p) + \operatorname{nullity}(M_p) = n + m. \tag{8}$$

Since $\operatorname{rank}(M_p) = m$, the $\operatorname{nullity}(M_p)$ is $\operatorname{nullity}(M_p) = n + m - m = n$. The basis of the null space of M_p is now $\{b_1,\dots,b_n\}$. Let F' be another PRF: $\mathscr{K} \times (\mathscr{P} \times [1,n]) \to \mathbb{F}_q$, where $\mathscr{P}$ denotes the domain of p's space. The following steps are used to generate the key k_p:

(i) $r_i \leftarrow F(k,p,i) \in \mathbb{F}_q$, $\forall i \in \{1,\dots,n\}$;

(ii) $k_p \leftarrow \sum_{i=1}^{n} r_i b_i \in \mathbb{F}_q^{n+m}$.

Let k_{repair} denote k_p (to distinguish with the notation k_p from the Keygen1). The Keygen2 is only executed and $\kappa' = \kappa + k_{\text{repair}}$ is given to a new server only if a repair phase happens. The key κ is already computed in the Keygen1 as a static information, and only k_{repair} is different in each repair time.

4.2. Encode

Step 1. Each client $\mathscr{C}_i$ $(i \in \{1,\dots,s\})$ computes g tags for g augmented blocks:

$$\forall i \in \{1,\dots,s\},\ \forall j \in \{1,\dots,g\} : t_{ij} = w_{ij} \cdot k_i. \tag{9}$$

Step 2. Each client $\mathscr{C}_i$ $(i \in \{1,\dots,s\})$ linearly combines the augmented blocks and the corresponding tags:

$\forall i \in \{1,\dots,s\}$:

(i) $\forall j \in \{1,\dots,g\}$, generate g coefficients: $\alpha_{ij} \xleftarrow{\text{rand}} \mathbb{F}_q$;

(ii) compute coded block: $w_{\mathscr{C}_i} = \sum_{j=1}^{g} \alpha_{ij} \cdot w_{ij}$;

(iii) compute tag: $t_{\mathscr{C}_i} = \sum_{j=1}^{g} \alpha_{ij} \cdot t_{ij}$.

Step 3. Each client $\mathscr{C}_i$ sends the pair of $(w_{\mathscr{C}_i}, t_{\mathscr{C}_i})$ to all h servers $(S_1,\dots,S_h)$. Each server S_x where $x \in \{1,\dots,h\}$ creates d pairs of coded block c_{xy} and corresponding tag t_{xy} where $y \in \{1,\dots,d\}$:

$\forall x \in \{1,\dots,h\}$, $y \in \{1,\dots,d\}$, S_x computes:

(i) $\forall i \in \{1,\dots,s\}$, generate s coefficients: $\beta_{xyi} \xleftarrow{\text{rand}} \mathbb{F}_q$;

(ii) compute coded block: $c_{xy} = \sum_{i=1}^{s} \beta_{xyi} \cdot w_{\mathscr{C}_i}$;

(iii) compute tag: $t_{xy} = \sum_{i=1}^{s} \beta_{xyi} \cdot t_{\mathscr{C}_i}$.

4.3. Check. The TPA is assigned the check responsibility. The TPA uses the key $\kappa = k_1 + \cdots + k_s$ to check h servers periodically. Note that the TPA is only given the sum κ without learning each component k_i where $i = \{1,\cdots,s\}$. Assume that the TPA does not collude with any server:

$\forall x \in \{1,\dots,h\}$:

(i) S_x computes:

(a) $\forall y \in \{1,\dots,d\}$; generate d coefficients $\gamma_{xy} \xleftarrow{\text{rand}} \mathbb{F}_q$;

(b) combine coded blocks: $c_x = \sum_{y=1}^{d} c_{xy} \cdot \gamma_{xy}$;

(c) combine tags: $t_x = \sum_{y=1}^{d} t_{xy} \cdot \gamma_{xy}$;

(ii) S_x sends $\{c_x, t_x\}$ to the TPA;

(iii) TPA computes $t'_x = c_x \cdot \kappa$;

(iv) TPA verifies: $t_x = t'_x$ $(*)$, and then returns true (this means that S_x is healthy), otherwise returns false.

Correctness of the Verification $(*)$

Consider

$$\begin{aligned} t_x &= \sum_{y=1}^{d} t_{xy} \cdot \gamma_{xy} \\ &= \sum_{y=1}^{d} \sum_{i=1}^{s} \gamma_{xy} \beta_{xyi} t_{\mathscr{C}_i} \\ &= \sum_{y=1}^{d} \sum_{i=1}^{s} \sum_{j=1}^{g} \gamma_{xy} \beta_{xyi} \alpha_{ij} t_{ij} \\ &= \sum_{y=1}^{d} \sum_{i=1}^{s} \sum_{j=1}^{g} \gamma_{xy} \beta_{xyi} \alpha_{ij} w_{ij} k_i, \\ t'_x &= c_x \cdot \kappa \\ &= \sum_{y=1}^{d} \gamma_{xy} c_{xy} \left(k_1 + \cdots + k_s\right) \\ &= \sum_{y=1}^{d} \sum_{i=1}^{s} \gamma_{xy} \beta_{xyi} w_{\mathscr{C}_i} \left(k_1 + \cdots + k_s\right) \\ &= \sum_{y=1}^{d} \sum_{i=1}^{s} \sum_{j=1}^{g} \gamma_{xy} \beta_{xyi} \alpha_{ij} w_{ij} \left(k_1 + \cdots + k_s\right). \end{aligned} \tag{10}$$

As described in Section 4.1.1 (Keygen 1), the property of k_p is that $\forall i \in \{1, \ldots, s\}, i \neq p, p \in \{1, \ldots, s\}, w_{ij} \cdot k_p = 0$. As a result, $t'_x = \sum_{y=1}^{d} \sum_{i=1}^{s} \sum_{j=1}^{g} \gamma_{xy} \beta_{xyi} \alpha_{ij} w_{ij} k_i$. Therefore, $t_x = t'_x$.

4.4. Repair. Suppose that the server S_r is detected as corrupted in the check phase. S_r is replaced by a new server S'_r. The server S'_r requires l healthy servers $S_{x_1}, \ldots, S_{x_l}$ to provide their combined packets consisting of the coded blocks and the tags. S'_r is given the key $\kappa' = \kappa + k_{\text{repair}}$, where k_{repair} is generated from the Keygen2, to check the provided packets.

Step 1. Each server S_x where $x \in \{x_1, \ldots, x_l\}$ linearly combines its d coded blocks and linearly combines its d tags. S_x then sends the aggregated coded block and aggregated tag to the new server S'_r:

$\forall x \in \{x_1, \ldots, x_l\}$, S_x performs:

(i) $\forall y \in \{1, \ldots, d\}$, generates d coefficients $\gamma_{xy} \xleftarrow{\text{rand}} \mathbb{F}_q$;

(ii) combine coded blocks: $c_x = \sum_{y=1}^{d} c_{xy} \cdot \gamma_{xy}$;

(iii) combine tags: $t_x = \sum_{y=1}^{d} t_{xy} \cdot \gamma_{xy}$;

(iv) send the package consisting of $\{c_x, t_x\}$ to S'_r.

Step 2. The new server S'_r checks whether each server S_x where $x \in \{x_1, \ldots, x_l\}$ provides a valid packet (pollution attack), using the key $\kappa' = (k_1 + \cdots + k_s) + k_{\text{repair}}$.

Given $\kappa' = (k_1 + \cdots + k_s) + k_{\text{repair}}$, S'_r computes:

(i) compute $t'_x = c_x \cdot \kappa'$;

(ii) check $t_x = t'_x$ $(**)$.

Step 3. The new server S'_r computes d coded blocks and d tags for itself:

$\forall y \in \{1, \ldots, d\}$, S'_r computes:

(i) $\forall x \in \{x_1, \ldots, x_l\}$, generate l coefficients $\theta_{xy} \xleftarrow{\text{rand}} \mathbb{F}_q$;

(ii) new coded blocks $c_{ry} = \sum_{x=x_1}^{x_l} c_x \cdot \theta_{xy}$;

(iii) new tags $t_{ry} = \sum_{x=x_1}^{x_l} t_x \cdot \theta_{xy}$.

Correctness of the Verification $(**)$ *in Step 2.* The way to prove the correctness of this verification is similar to the correctness of the verification $(*)$ in the check phase. The only different thing is that not only $k_1, \ldots, k_s$ but also k_{repair} participates in combining the coded blocks and homomorphic tags. As described in the Keygen2 (Section 4.1.2), $\forall i \in \{1, \ldots, s\}$, $p \xleftarrow{\text{rand}} \mathbb{F}_q$, $p > s$, and we have $w_{ij} \cdot k_p = 0$.

5. Security Analysis

5.1. Security against Mobile Adversaries. To prevent mobile adversaries, a data repair threshold is given as follows.

Theorem 1. *The original files $F_1, \ldots, F_s$ of the clients can be recovered if in any epoch, at least l out of h servers collectively store $m = sg$ coded blocks which are linearly independent combinations of m original file blocks; and the matrix consisting of the accumulated coefficients has full rank (i.e., rank m).*

Proof. Each server S_x where $x \in \{1, \ldots, h\}$ contains d coded blocks: $\{c_{xy}\}$ $(y \in \{1, \ldots, d\})$. Each coded block c_{xy} is computed from $m = sg$ augmented blocks w_{ij} $(i \in \{1, \ldots, s\}, j \in \{1, \ldots, g\})$ as $c_{xy} = \sum_{i=1}^{s} \sum_{j=1}^{g} \beta_{xyi} \cdot \alpha_{ij} \cdot w_{ij}$. To recover the original files, m augmented blocks $(w_{11}, \ldots, w_{1g}, \ldots, w_{s1}, \ldots, w_{sg})$ are viewed as the variables that need to be solved. To solve such m variables, at least m coded blocks are needed such that the coefficient matrix has full rank because the number of variables in an equation system has to be less than or equal to the number of independent equations:

$$\begin{aligned} c_{xy_1} &= \sum_{i=1}^{s}\sum_{j=1}^{g}\beta_{xyi_1}\cdot\alpha_{ij_1}\cdot w_{ij} \\ c_{xy_2} &= \sum_{i=1}^{s}\sum_{j=1}^{g}\beta_{xyi_2}\cdot\alpha_{ij_2}\cdot w_{ij} \\ &\vdots \\ c_{xy_m} &= \sum_{i=1}^{s}\sum_{j=1}^{g}\beta_{xyi_m}\cdot\alpha_{ij_m}\cdot w_{ij}. \end{aligned} \tag{11}$$

Therefore, at least l servers which collectively store $m = s \cdot g$ coded blocks in each epoch are required. $\lceil m/d \rceil \le l < h$. □

5.2. Security against Curious Adversaries. The following theorem gives the probability of the adversary to recover the secret keys and shows that the probability is negligible.

Theorem 2. *The secret keys of the clients are secured from the TPA and the new server.*

Proof. The TPA checks h servers $(S_1, \ldots, S_h)$ in the check phase using the key $\kappa = k_1 + \cdots + k_s$. Similarly, the new servers S'_r check l healthy servers in the repair phase using the key $\kappa' = (k_1 + \cdots + k_s) + k_{\text{repair}}$. The problem of security is now the problem of solving s variables (in the case of the TPA) and $s+1$ variables (in the case of S'_r) given one equation. The only method to solve these variables is to try all possible variable sets and test whether they satisfy this equation by using trial-and-error method with brute-force search. Let $\mathscr{K}$ denote the key space. Each k_i $(i \in \{1, \ldots, s\})$, k_{repair}, κ, and κ' belong to the finite field $\mathbb{F}_q^{n+m}$ (which has $(n+m)\log_2 q$ bit-length), and therefore $\mathscr{K} = q^{n+m}$. The number of testing times is $(\mathscr{K})^{s-1}$ in the case of the TPA and $(\mathscr{K})^{s}$ in the case of S'_r. Therefore, the probability for choosing s variables is $1/q^{(n+m)(s-1)}$ in the case of the TPA and the probability for choosing $s+1$ variables is $1/q^{(n+m)s}$ in the case of S'_r. If q is chosen as a large prime (e.g., 160 bits), $k_1, \ldots, k_s$, and k_{repair} cannot be solved in a polynomial time. Ergo, the probability of TPA and S'_r are negligible. □

5.3. Security against Response Forgeries. After controlling S_x, suppose that, in the check phase, the adversary $\mathscr{A}$ sends a pair of forged coded block and forged tag (c''_x, t''_x) to the TPA, instead of a valid pair of (c_x, t_x).

Theorem 3. *The advantage of a forgery adversary to pass the check phase is*

$$\mathsf{Adv}_{\mathscr{A}}(\mathsf{verify}) = \mathsf{Adv}_{\mathscr{A}}(\mathsf{PRF}) + \frac{1}{q^{(n+m)s}}. \tag{12}$$

Proof. To be able to generate (c''_x, t''_x) which holds the verification $t''_x = c''_x \cdot \kappa$, the adversary $\mathscr{A}$ has to obtain κ. Since the TPA is assumed not to collude with any server and κ is sent to $\mathscr{A}$ though a secure channel, a possible way for $\mathscr{A}$ is to attack the Keygen1 in which the key k_p of $\mathscr{C}_p$ $(p \in \{1, \ldots, s\})$ is computed as

(i) $r_i \leftarrow F(k, p, i) \in \mathbb{F}_q$, $\forall i \in \{1, \ldots, n+g\}$;

(ii) $k_p \leftarrow \sum_{i=1}^{n+g} r_i b_i \in \mathbb{F}_q^{n+m}$.

The advantage of $\mathscr{A}$ on r_i is $\mathsf{Adv}_{\mathscr{A}}(\mathsf{PRF})$. Since $k_p \in \mathbb{F}_q^{n+m}$, the advantage of $\mathscr{A}$ on k_p is $1/q^{n+m}$. The advantage of $\mathscr{A}$ on $\kappa = \sum_{i=1}^{s} k_i$ is $1/q^{(n+m)s}$. Therefore, $\mathsf{Adv}_{\mathscr{A}}(\mathsf{verify}) = \mathsf{Adv}_{\mathscr{A}}(\mathsf{PRF}) + 1/q^{(n+m)s}$. If F is unforgeable and q is chosen large enough, for example, 160 bits, the advantage of $\mathscr{A}$ is negligible: $\mathsf{Adv}_{\mathscr{A}}(\mathsf{verify}) < \epsilon$. □

5.4. Security against Pollution Attack. Suppose that the server S_r is checked as a corrupted server and $S_{x_1}, \ldots, S_{x_l}$ are checked as healthy servers in the check phase. Then, $S_{x_1}, \ldots, S_{x_l}$ are required to repair S_r by providing their coded blocks and tags to the new server S'_r. In the repair phase, the adversary $\mathscr{A}$ attacks S_{x_p} $(x_p \in \{x_1, \ldots, x_l\})$ and then provides an invalid packet to the new server S'_r (pollution attack). Similar to Theorem 2, the advantage of $\mathscr{A}$ to pass the pollution attack check (Step 2 in the repair phase) is

$$\mathsf{Adv}_{\mathscr{A}}(\mathsf{pollution}) = \mathsf{Adv}_{\mathscr{A}}(\mathsf{PRF}) + \frac{1}{q^{(n+m)(s+1)}}. \tag{13}$$

The different thing is that the advantage of $\mathscr{A}$ on $\kappa' = k_{\text{repair}} + \sum_{i=1}^{s} k_i$ is $1/q^{(n+m)(s+1)}$, not $1/q^{(n+m)s}$ as Theorem 2 because the adversary does not own κ'.

We also consider a stronger adversary $\mathscr{A}$ who attacks S'_r right after the repair phase to steal κ' from S'_r. $\mathscr{A}$ then uses κ' to pass pollution attack check in another later repair phases. However, since k_{repair} is different in each repair time as explained in the Keygen2 (Section 4.1.2), the advantage for $\mathscr{A}$ to guess k_{repair} is $\mathsf{Adv}_{\mathscr{A}}(\mathsf{PRF}) + 1/q^{(n+m)}$.

6. Efficiency Analysis

Table 1 compares the features and efficiency of the proposed MD-POR scheme with some previous schemes. The RDC-NC [12] and NC-Audit [15] schemes are chosen for the comparison because they have the same scenario as the MD-POR scheme at most. One notable thing is that because the RDC-NC and NC-Audit schemes only consider a single client unlike the MD-POR scheme, we assume that s clients participate in the RDC-NC and NC-Audit schemes so that the comparisons are fair. However, these s clients in the RDC-NC and NC-Audit schemes can only perform in parallel instead of simultaneously combination as the MD-POR scheme. That parameter s in the RDC-NC and NC-Audit schemes does not affect the checking and repairing complexity because only one client can check and repair the servers. That s only affects the storage cost on server-side and the communication cost of the encode phase in the RDC-NC and NC-Audit schemes.

Table 1: Comparison.

		RDC-NC [12]	NC-Audit [15]	MD-POR (our scheme)
Feature	Multiclient	No	No	Yes
	Direct repair	No	Not completed	Yes
	Symmetric key	Yes	Yes	Yes
	public authentication	No	Yes	Yes
Storage complexity	Client-side	$O(5(n+g)\log_2 q)$	$O((n+g)\log_2 q)$	$O((n+sg)\log_2 q)$
	Server-side	$O(sdh((\|F\|/g)+g))$	$O(sdh((\|F\|/g)+g))$	$O(dh((\|F\|/g)+sg))$
	TPA-side	N/A	$O((n+g+gdh)\log_2 q)$	$O((n+sg)\log_2 q)$.
Encoding complexity	Computation (client)	$O(gdh)$	$O(gdh)$	$O(g)$
	Computation (server)	$O(1)$	$O(1)$	$O(sdh)$
	Computation (TPA)	N/A	$O(1)$	$O(1)$
	Communication	$O(sdh((\|F\|/g)+g))$	$O(sdh((\|F\|/g)+g)+sgdh)$	$O(hs((\|F\|/g)+sg))$
Checking complexity	Computation (client)	$O(h)$	$O(1)$	$O(1)$
	Computation (server)	$O(hd)$	$O(hd)$	$O(hd)$
	Computation (TPA)	N/A	$O(h)$	$O(h)$
	Communication	$O(h((\|F\|/g)+g))$	$O(h((\|F\|/g)+g))$	$O(h((\|F\|/g)+sg))$
Repairing complexity	Computation (client)	$O((l+1)d)$	$O(1)$	$O(1)$
	Computation (server)	$O(dl)$	$O(dl)$	$O(dl)$
	Computation (new server)	N/A	$O(dl)$	$O(dl)$
	Computation (TPA)	N/A	$O(l)$	$O(1)$
	Communication	$O((l+d)((\|F\|/g)+g))$	$O(l((\|F\|/g)+g)+ld)$	$O(l((\|F\|/g)+sg))$

6.1. Storage Cost

6.1.1. Client-Side. In the RDC-NC scheme, because the client keeps five secret keys in $\mathbb{F}_q^{n+g}$, the client storage is $O(5(n+g)\log_2 q)$. In the NC-Audit scheme, because the client keeps only one secret key in $\mathbb{F}_q^{n+g}$, the client storage is $O((n+g)\log_2 q)$. Meanwhile, the MD-POR scheme has s keys for s clients, each in $\mathbb{F}_q^{n+sg}$, and thus the storage cost per client is $O((n+sg)\log_2 q)$.

6.1.2. Server-Side. The size of a file block is $|v| = |F|/g$. The form of an augmented block is $w_i = (v_i, \overbrace{\underbrace{0,\ldots,0,1}_{i},0,\ldots,0}^{m=sg})$ as indicated in Section 2.3. In the RDC-NC and NC-Audit schemes, since $s = 1$, the size of an augmented block is $|w| = |F|/g + g$. In the MD-POR scheme, since $s = g$, the size of an augmented block is $|w| = |F|/g + sg$. Furthermore, the size of a coded block is $|c| = |w|$ because each coded block is a linear combination of augmented blocks. The number of servers is h. Each server stores d coded blocks. s clients are assumed to participate in the RDC-NC and NC-Audit schemes in parallel. Therefore, the server storage in the RDC-NC and NC-Audit schemes is $O(sdh(|F|/g + g))$. The server storage in the MD-POR scheme is $O(dh(|F|/g + sg))$.

6.1.3. TPA-Side. The RDC-NC scheme does not have a TPA. In the NC-Audit scheme, the TPA not only keeps a key in $\mathbb{F}_q^{n+g}$ for verification (which is $O((n+g)\log_2 q)$) but also stores the coding coefficients in $\mathbb{F}_q$ which are used to compute all coded blocks (which is $O(gdh\log_2 q)$). Hence, the total TPA storage in the NC-Audit scheme is $O((n+g+gdh)\log_2 q)$. In the MD-POR scheme, the TPA is given $\kappa = k_1 + \cdots + k_s \in \mathbb{F}_q^{n+m}$ (Section 4.1.1). In other words, $\kappa \in \mathbb{F}_q^{n+sg}$. The TPA storage in the MD-POR scheme is thus $O((n+sg)\log_2 q)$.

6.2. Encoding Cost

6.2.1. Computation on Client-Side. In the RDC-NC and NC-Audit schemes, during the encode phase, each client combines g augmented blocks (which is $O(g)$) to create dh coded blocks in order to store d coded blocks in each of h servers. The cost in these schemes is thus $O(gdh)$. In the MD-POR scheme, each client only needs to combine g augmented blocks (which is $O(g)$) and distributes the result to all the servers. The servers will create coded blocks by themselves. The cost in the MD-POR is thus $O(g)$.

6.2.2. Computation on Server-Side. In the RDC-NC and NC-Audit schemes, the servers do not need to do anything and only need to receive the coded blocks computed by the clients. The cost in these schemes is thus $O(1)$. In the MD-POR scheme, each of h servers combines s coded blocks from the clients and computes d coded blocks for itself. The cost in the MD-POR is thus $O(sdh)$.

6.2.3. Computation on TPA-Side. In the RDC-NC scheme, the TPA does not exist. In the NC-Audit and MD-POR schemes, the TPA does nothing during the encode phase; and the costs are thus $O(1)$.

6.2.4. Communication. In the RDC-NC scheme, the client creates dh coded blocks and sends d coded blocks to each of h servers. The size of a coded block in these scheme is $(|F|/g+g)$ as mentioned in Section 6.1.2. The number of clients is s. Therefore, the communication cost is $O(sdh(|F|/g+g))$. In the NC-Audit scheme, the communication is also similar to the RDC-NC scheme. However, the difference is that the client in the NC-Audit scheme not only sends the coded blocks to the servers, but also sends all $sgdh$ coefficients which are used to create the coded blocks to the servers. The cost in the NC-Audit scheme is thus $O(sdh(|F|/g+g)+sgdh)$. In the MD-POR scheme, each of s clients sends the aggregated coded block to each of h servers. The size of a coded block in the MD-POR scheme is $(|F|/g+sg)$ (see (5)). The cost in the MD-POR scheme is thus $O(hs(|F|/g+sg))$.

6.3. Checking Cost

6.3.1. Computation on Client-Side. In the RDC-NC scheme, the client receives the aggregated coded block from each of h servers and verifies each of them using his/her secret key; the cost is thus $O(h)$. In the NC-Audit and MD-POR schemes, the TPA will check the servers instead of the client. The cost in the NC-Audit and MD-POR schemes is thus $O(1)$ on the client-side.

6.3.2. Computation on Server-Side. In all three schemes, each of h servers combines its d coded blocks to send the result (an aggregated coded block) back to the verifier. The verifier is the client in the case of the RDC-NC scheme and is the TPA in the case of the NC-Audit and MD-POR schemes. The cost in all three schemes is $O(hd)$.

6.3.3. Computation on TPA-Side. In the RDC-NC scheme, the TPA does not exist. In the NC-Audit and MD-POR schemes, the TPA verifies the aggregated coded block which is accommodated from each of h servers. Each verification only takes one operation. The cost in the NC-Audit and MD-POR schemes is $O(h)$.

6.3.4. Communication. In the RDC-NC and NC-Audit schemes, during the check phase, each of h servers sends its aggregated coded block to the client. The size of that coded block is $(|F|/g+g)$. The cost in these schemes is thus $O(h(|F|/g+g))$. In the MD-POR scheme, the mechanism is the same as the RDC-NC and NC-Audit scheme, but the different thing is that the size of a coded block in the MD-POR scheme is $(|F|/g+sg)$. The cost in the MD-POR scheme is thus $O(h(|F|/g+sg))$.

6.4. Repairing Cost

6.4.1. Computation on Client-Side. In the RDC-NC scheme, in the repair phase, the client firstly has to check pollution attack in l coded blocks which are provided from l healthy servers (which is $O(l)$). Thereafter, the client computes d new coded blocks for the new server by combining l provided coded blocks (which is $O(ld)$). Hence, the computation cost on the client-side in the RDC-NC scheme is $O((l+1)d)$. In the NC-Audit and MD-POR schemes, the client(s) does nothing.

6.4.2. Computation on Server-Side. In the RDC-NC scheme, each of l healthy servers is required to combine its d coded blocks. Therefore, the computation cost on the server-side is $O(dl)$. The cost in the new server is N/A because the direct repair feature is not supported in the RDC-NC scheme. In the NC-Audit and MD-POR schemes, not only l healthy servers combine their coded blocks (which is $O(dl)$) but also the new server computes its d new coded blocks by combining l provided coded blocks (which is $O(dl)$).

6.4.3. Computation on TPA-Side. The RDC-NC scheme does not have a TPA. In the NC-Audit scheme, the TPA has to check pollution attack in l provided coded blocks (which is $O(l)$). In the MD-POR scheme, the TPA does nothing because the new server will check pollution attack, not the TPA as the NC-Audit scheme. Therefore, the computation cost on the TPA-side in the MD-POR scheme is $O(1)$.

6.4.4. Communication. In the RDC-NC scheme, each of l healthy servers sends an aggregated coded block whose size is $|F|/g+g$ to the client (which is $O(l(|F|/g+g))$). After computing d new coded blocks, the client sends them to the new server (which is $O(d(|F|/g+g))$). As a result, the communication cost in the RDC-NC scheme is $O((l+d)(|F|/g+g))$. In the NC-Audit scheme, each of l healthy servers also sends an aggregated coded block to the new server (which is $O(l(|F|/g+g))$). Then, the new server sends its linear coefficients which are used to compute d new coded blocks from l provided coded blocks to the TPA (which is $O(ld)$). Therefore, the communication cost in the NC-Audit scheme is $O(l(|F|/g+g)+ld)$. In the MD-POR scheme, only each of l healthy servers sends an aggregated coded block to the new server (each coded block has the size $|F|/g+sg$). Therefore, the communication cost in the MD-POR scheme is $O(l(|F|/g+sg))$.

In summary, although the MD-POR scheme supports many heavy features, its cost of the whole scheme is still better than the previous schemes. Let $O_p(A)$, $O_p(B)$, and $O_p(C)$ denote the whole computation costs of the RDC-NC, NC-Audit, and MD-POR schemes, respectively. Let $O_m(A)$, $O_m(B)$, and $O_m(C)$ denote the whole communication costs of the RDC-NC, NC-Audit, and MD-POR schemes, respectively. Let $O_s(A)$, $O_s(B)$, and $O_s(C)$ denote the whole storage costs of the RDC-NC, NC-Audit, and MD-POR schemes, respectively. In reality, d and g are far larger than s and h $(d, g \gg s, h)$, $l \in \{1, \ldots, h\}$, and $d > g$. From Table 1, the following results are obtained. $O_p(A) - O_p(C) = (gdh+d) - (sdh+g) > 0$ because $g \gg s$. $O_p(B) - O_p(C) = (gdh+l) - (sdh+g) > 0$ because $g \gg s$. $O_m(A) - O_m(C) = (dhs+d-hs)(|F|/g) + g(sdh+h+l+d-hs^2-hs-ls) > 0$ because $d \gg s$ and $1 \leq l \leq h$. $O_m(B) - O_m(C) = (dhs-hs)(|F|/g) + g(2sdh+h+l-hs^2-hs-ls) + ld > 0$ because $d \gg s$ and $1 \leq l \leq h$. $O_s(A) - O_s(C) = (3n+5g)\log_2 q + (s-1)dh(|F|/g) - 2sg\log_2 q > 0$ because $|F|/g = n\log_2 q$ and $d > g$. $O_s(B) - O_s(C) = (s-1)dh(|F|/g) + g\log_2 q(dh-2s+2) > 0$ because $d \gg s$.

7. Performance Evaluation

This section evaluates the computation and communication performances of the proposed MD-POR scheme to show that it is applicable for a real system. A program written by Python 2.7.3 is executed using a computer with Intel Core i5 processor, 2.4 GHz, 4 GB of RAM, and Windows 7 64-bit OS. The length of the prime q is set to be 160 bits. The number of clients is set to be 5 ($s = 5$) which is also the parameter used in the performance evaluation of the InterMac in the paper [16]. The number of servers is set to be 10 ($h = 10$). The number of coded blocks stored in each server is set to be 100 ($d = 100$). The number of healthy servers which are used for repairing is set to be 3 ($l = 3$). The size of each file block is set to be 2^{23} bits (1MB). Each result is the average of 100 runs.

The experiment results are observed with three sets of computation performance and a set of communication performance by varying the file size of each client. The computation results are depicted in Figure 4 (encode), Figure 5 (check), and Figure 6 (repair). The communication result is depicted in Figure 7 (encode, check, and repair).

Computation Performance. The experiment results reveal that the computation time increases almost linearly as the file size increases, and each graph has a different slope. Only the computation time of TPA-side in the check phase is almost constant. In the encode phase, the slopes of increment in the graphs of client-side and server-side are approximately 0.04 and 0.002, respectively. Therefore, if the file size is 1 GB, the computation time on client-side and server-side is estimated as 41 seconds and 2 seconds, respectively. Note that the encode phase only is executed one time in the beginning; meanwhile, the check phase is executed many times during system lifetime and the repair phase is executed once a corruption is detected in the check phase. Consequently, the check and repair phases are more important than the encode phase. In the check phase, the slopes of increment in the graphs of server-side and TPA-side are approximately 0.0005 and 0, respectively. Therefore, if the file size is 1 GB, the computation time on server-side and TPA-side is estimated as 0.52 seconds and 0.02 seconds, respectively. Similarly, in the repair phase, the slopes of increment in the graphs of healthy server-side and new server-side are approximately 0.0005 and 0.0014, respectively. Therefore, if the file size is 1 GB, the computation time on healthy server-side and new server-side is estimated as 0.52 seconds and 1.47 seconds, respectively.

Communication Performance. The MD-POR scheme is performed with the bandwidth of 300 Mbps. The experiment results reveal that the communication time increases almost linearly as the file size increases, and each graph in Figure 7 has a different slope. The slopes of increment in the graphs of the encode phase, the check phase, and the repair phase are approximately 0.048, 0.008, and 0.006, respectively. Therefore, if the file size is 1 GB, the communication time of the encode phase, check phase, and repair phase is estimated as 49.27 seconds, 7.86 seconds, and 5.83 seconds, respectively. In addition, the size of the response from each server is given as follows. The response size of 50 MB, 75 MB, 100 MB, 125 MB,

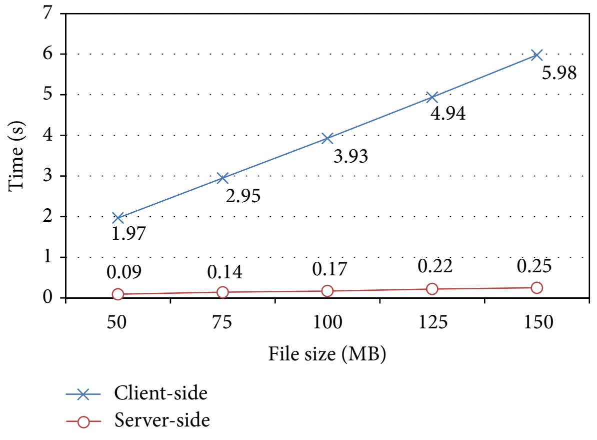


Figure 4: The computation time performance of the encode algorithm.

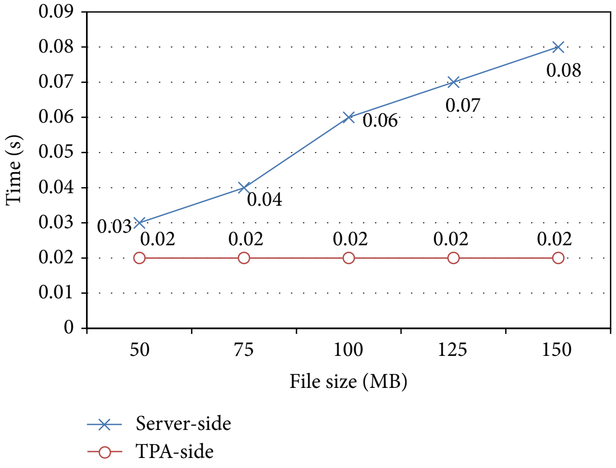


Figure 5: The computation time performance of the check algorithm.

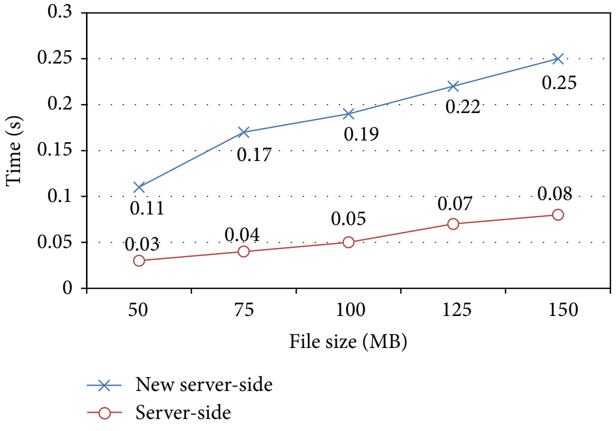


Figure 6: The computation time performance of the repair algorithm.

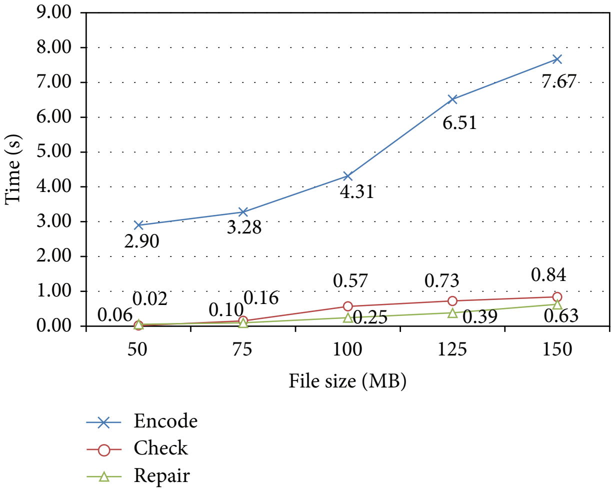


FIGURE 7: The communication time performance.

and 150 MB file size is 13 KB, 19 KB, 26 KB, 32 KB, and 38 KB, respectively. Therefore, if the file size is 1 GB, the response size is estimated as 264.87 KB.

The above results indicate that the computation and communication performances are very fast even when the file size is 1 GB.

8. Conclusion and Future Work

In this paper, a network coding-based POR scheme named MD-POR has been proposed. The MD-POR scheme supports multiclient, symmetric key-based direct repair and public authentication features. Moreover, the MD-POR scheme can protect against a strong adversary who can perform mobile attack, curious attack, response forgery, and pollution attack. Furthermore, the efficiency analysis based on the complexity theory shows that although the MD-POR scheme supports many features, its costs are not bad compared with the previous schemes. The experiment results reveal that the computation time increases as the file size increases. However, the graphs show that the slope of increment for the MD-POR scheme increases merely. Future work is invested to implement two previous RDC-NC and NC-Audit schemes in order to compare with the MD-POR scheme. This paper have implemented only the MD-POR scheme to show that its computation cost is applicable for a real system.

Appendix

How the InterMac and the Network Coding Combine Together

In this appendix, an example is given to explain how the InterMac and the network coding work together to compute the keys. Suppose that there are two clients: $\mathscr{C}_1$ and $\mathscr{C}_2$. The augmented blocks are $w_{11} = (2, 1, 0, 0, 0)$, $w_{12} = (3, 0, 1, 0, 0)$, $w_{21} = (1, 0, 0, 1, 0)$, and $w_{22} = (5, 0, 0, 0, 1)$. All operations work in $\mathbb{F}_7$.

The Key of $\mathscr{C}_1$. A matrix M_1 is constructed in a way that it consists of all augmented blocks which do not belong to $\mathscr{C}_1$:

$$M_1 = \begin{pmatrix} w_{21} \\ w_{22} \end{pmatrix} = \begin{pmatrix} 1, 0, 0, 1, 0 \\ 5, 0, 0, 0, 1 \end{pmatrix}. \tag{A.1}$$

M_1 is then reduced by the Gauss-Jordan elimination to a row echelon form as follows:

$$M_1' = \begin{pmatrix} \boxed{1}, 0, 0, 1, 0 \\ \\ 0, 0, 0, \boxed{1}, 4 \end{pmatrix}. \tag{A.2}$$

Let $\delta_1, \ldots, \delta_5$ denote the unknown variables which correspond to the columns of M_1'. Let $\delta = [\delta_1, \ldots, \delta_5]^T$. The pivots which are the values in the squares belong to δ_1 and δ_4. The free variables are δ_2, δ_3 and δ_5. Solving $M_1' \cdot \delta = 0$, we have $\delta_1 + \delta_4 = 0$ and $\delta_4 + 4\delta_5 = 0$. Let $\delta_2 = a, \delta_3 = b$, and $\delta_5 = c$. One has

$$\begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \\ \delta_5 \end{pmatrix} = a \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + b \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + c \begin{pmatrix} 4 \\ 0 \\ 0 \\ -4 \\ 1 \end{pmatrix}. \tag{A.3}$$

Because the number of free variables is 3, the number of elements in the basis is also 3. Namely, the basis is as follows:

$$\left\{ \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \\ 0 \\ -4 \\ 1 \end{pmatrix} \right\}. \tag{A.4}$$

Suppose that the random values are 2, 3, and 2. The key of $\mathscr{C}_1$ is computed as

$$\begin{aligned} k_1 &= 2 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 3 \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \\ &\quad + 2 \begin{pmatrix} 4 \\ 0 \\ 0 \\ -4 \\ 1 \end{pmatrix} \pmod 7 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 6 \\ 2 \end{pmatrix}. \end{aligned} \tag{A.5}$$

Here observe that k_1 is orthogonal to w_{21} and w_{22}. In other words, $k_1 \cdot w_{21} = k_1 \cdot w_{22} = 0$:

$$\begin{aligned} &\begin{pmatrix} 1 \\ 2 \\ 3 \\ 6 \\ 2 \end{pmatrix} (1, 0, 0, 1, 0) \pmod 7 \\ &\quad = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 6 \\ 2 \end{pmatrix} (5, 0, 0, 0, 1) \pmod 7 = 0. \end{aligned} \tag{A.6}$$

The key of $\mathscr{C}_2$. Similarly, k_2 is also constructed in the same way as k_1

$$M_2 = \begin{pmatrix} w_{11} \\ w_{12} \end{pmatrix} = \begin{pmatrix} 2,1,0,0,0 \\ 3,0,1,0,0 \end{pmatrix} \xrightarrow{\text{row-ech.}} M_2' = \begin{pmatrix} \boxed{1},4,0,0,0 \\ 0,\boxed{1},4,0,0 \end{pmatrix}. \tag{A.7}$$

The free variables are δ_3, δ_4, and δ_5. The basis is as follows:

$$\left\{ \begin{pmatrix} 16 \\ -4 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\}. \tag{A.8}$$

Suppose that the random values are 2, 2, and 1. The key k_2 is constructed as

$$k_2 = 2\begin{pmatrix} 16 \\ -4 \\ 1 \\ 0 \\ 0 \end{pmatrix} + 2\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + 1\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \pmod 7 = \begin{pmatrix} 4 \\ 6 \\ 2 \\ 2 \\ 1 \end{pmatrix}. \tag{A.9}$$

k_2 is orthogonal to w_{11} and w_{12}: $k_2 \cdot w_{11} = k_2 \cdot w_{12} = 0$.

The Key of the TPA. The TPA is given the key κ as follows:

$$\kappa = k_1 + k_2 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 6 \\ 2 \end{pmatrix} + \begin{pmatrix} 4 \\ 6 \\ 2 \\ 2 \\ 1 \end{pmatrix} \pmod 7 = \begin{pmatrix} 5 \\ 1 \\ 5 \\ 1 \\ 3 \end{pmatrix}. \tag{A.10}$$

The Key of the New Server. The new server is given the key κ' as $\kappa' = \kappa + k_{\text{repair}}$ where k_{repair} is constructed as follows. Firstly, a matrix M_r is constructed in a way that it consists of all augmented blocks:

$$M_r = \begin{pmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{pmatrix} = \begin{pmatrix} 2,1,0,0,0 \\ 3,0,1,0,0 \\ 1,0,0,1,0 \\ 5,0,0,0,1 \end{pmatrix}. \tag{A.11}$$

M_r is then reduced by the Gauss-Jordan elimination to a row echelon form as follows:

$$M_r' = \begin{pmatrix} \boxed{1},0,0,0,3 \\ 0,\boxed{1},0,0,1 \\ 0,0,\boxed{1},4,0 \\ 0,0,0,\boxed{1},4 \end{pmatrix}. \tag{A.12}$$

The free variable is δ_5. The basis is as follows:

$$\left\{ \begin{pmatrix} -3 \\ -1 \\ 16 \\ -4 \\ 1 \end{pmatrix} \right\}. \tag{A.13}$$

Suppose the random value is 3. k_{repair} is computed as

$$k_{\text{repair}} = 3\begin{pmatrix} -3 \\ -1 \\ 16 \\ -4 \\ 1 \end{pmatrix} \pmod 7 = \begin{pmatrix} 5 \\ 4 \\ 6 \\ 2 \\ 3 \end{pmatrix}. \tag{A.14}$$

k_{repair} is orthogonal to all augmented blocks: $k_{\text{repair}} \cdot w_{11} = k_{\text{repair}} \cdot w_{12} = k_{\text{repair}} \cdot w_{21} = k_{\text{repair}} \cdot w_{22}$. Then, κ' is computed as

$$\kappa' = \kappa + k_{\text{repair}} = \begin{pmatrix} 5 \\ 1 \\ 5 \\ 1 \\ 3 \end{pmatrix} + \begin{pmatrix} 5 \\ 4 \\ 6 \\ 2 \\ 3 \end{pmatrix} \pmod 7 = \begin{pmatrix} 3 \\ 5 \\ 4 \\ 3 \\ 6 \end{pmatrix}. \tag{A.15}$$

Notations

s:	number of clients
$\mathscr{C}_i$:	client ($i \in \{1, \ldots, s\}$)
k_i:	secret key of $\mathscr{C}_i$
F_i:	original file of $\mathscr{C}_i$
g:	number of file blocks in F_i of each client (g is the same in all clients)
i:	client index ($i \in \{1, \ldots, s\}$)
j:	file block index ($j \in \{1, \ldots, g\}$)
v_{ij}:	file block ($i \in \{1, \ldots, s\}, j \in \{1, \ldots, g\}$)
w_{ij}:	augmented block of v_{ij} ($i \in \{1, \ldots, s\}, j \in \{1, \ldots, g\}$)
$\mathbb{F}_q^n$:	a n-dimensional vector space over a finite field $\mathbb{F}_q$ where q denotes a large prime
m:	$m = s \cdot g$
h:	number of servers
l:	number of healthy servers which are used to repair a corrupted server
d:	number of coded blocks in each server
x:	server index ($x \in \{1, \ldots, h\}$)
y:	coded block index in each server ($y \in \{1, \ldots, d\}$)
S_x:	server ($x \in \{1, \ldots, h\}$)
c_{xy}:	coded block ($x \in \{1, \ldots, h\}, y \in \{1, \ldots, d\}$)
t_{xy}:	tag of c_{xy} ($x \in \{1, \ldots, h\}, y \in \{1, \ldots, d\}$)
TPA:	third party auditor
κ:	key of the TPA
κ':	key of the new server
$\mathscr{A}$:	adversary.

Disclosure

The preliminary version of this paper was presented at WISA 2014 [24].

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This study was partly supported by Grant-in-Aid for Young Scientists (B) (25730083) and CREST, JST.

References

[1] A. Juels and B. S. Kaliski Jr., "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 584–597, November 2007.

[2] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of the 4th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '08)*, pp. 90–107, Melbourne, Australia, December 2008.

[3] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: theory and implementation," in *Proceedings of the Workshop on Cloud Computing Security (CCSW '09)*, pp. 43–54, 2009.

[4] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: multiple-replica provable data possession," in *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS '08)*, pp. 411–420, July 2008.

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

[6] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[7] T. Ho, M. Medard, R. Koetter et al., "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[8] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[9] S. Agrawa and D. Boneh, "Homomorphic MACs: MAC-based integrity for network coding," in *Proceedings of the 7th Conference on Applied Cryptography and Network Security (ACNS '09)*, pp. 292–305, 2009.

[10] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[11] J. Li, S. Yang, X. Wang, X. Xue, and B. Li, "Tree-structured data regeneration in distributed storage systems with network coding," in *Proceedings of the 29th Conference on Information Communications (INFOCOM '10)*, pp. 2892–2900, 2010.

[12] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proceedings of the ACM Workshop on Cloud Computing Security Workshop (CCSW '10)*, pp. 31–42, 2010.

[13] N. Cao, S. Yu, Z. Yang, W. Lou, and Y. T. Hou, "LT codes-based secure and reliable cloud storage service," in *Proceedings of the 31st IEEE Conference on Computer Communications (INFOCOM '12)*, pp. 693–701, March 2012.

[14] H. C. H. Chen, Y. Hu, P. P. C. Lee, and Y. Tang, "NCCloud: applying network coding for the storage repair in a cloud-of-clouds," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST '12)*, p. 21, 2012.

[15] A. Le and A. Markopoulou, "NC-audit: auditing for network coding storage," in *Proceedings of the International Symposium on Network Coding (NetCod '12)*, pp. 155–160, 2012.

[16] A. Le and A. Markopoulou, "On detecting pollution attacks in inter-session network coding," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '12)*, pp. 343–351, Orlando, Fla, USA, March 2012.

[17] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 847–859, 2011.

[18] Y. Zhu, H. Wang, Z. Hu, G. J. Ahn, H. Hu, and S. S. Yau, "Dynamic audit services for integrity verification of outsourced storages in clouds," in *Proceedings of the 26th Annual ACM Symposium on Applied Computing (SAC '11)*, pp. 1550–1557, March 2011.

[19] Y. Yu, Y. Mu, J. Ni, J. Deng, and K. Huang, "Identity privacy-preserving public auditing with dynamic group for secure mobile cloud storage," in *Proceedings of the 8th International Conference on Network and System Security (NSS '14)*, pp. 28–40, Xi'an, China, October 2014.

[20] R. Koetter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, 2003.

[21] S. Agrawal, D. Boneh, X. Boyen, and D. M. Freeman, "Preventing pollution attacks in multi-source network coding," in *Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography (PKC '10)*, pp. 161–176, 2010.

[22] L. Czap and I. Vajda, "Signatures for multi-source network coding," IACR Cryptology ePrint Archive 2010/328, 2010.

[23] W. Yan, M. Yang, L. Li, and H. Fang, "Short signature scheme for multi-source network coding," *Computer Communications*, vol. 35, no. 3, pp. 344–351, 2012.

[24] K. Omote and T. P. Thao, "MDNC: multi-source and direct repair in network coding-based proof of retrievability scheme," in *Proceedings of the 15th International Workshop on Information Security Applications (WISA '14)*, pp. 177–188, 2014.

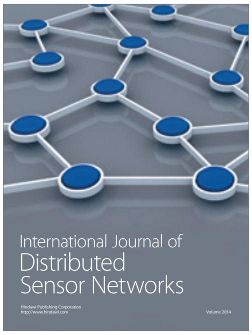
The Scientific World Journal

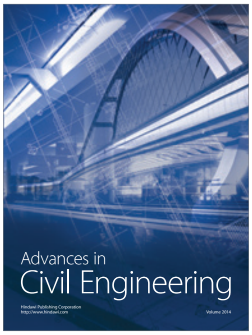
International Journal of
Rotating Machinery

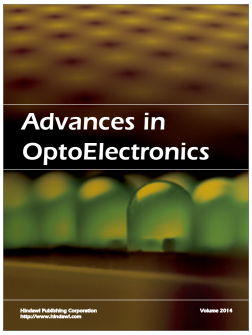
Journal of
Engineering

Journal of
Sensors

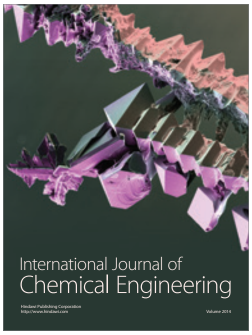
Advances in
Mechanical Engineering


International Journal of
Distributed Sensor Networks


Advances in
Civil Engineering

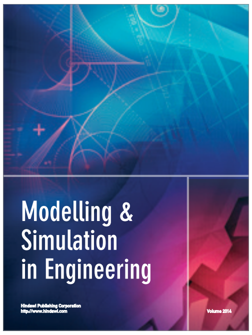

Advances in
OptoElectronics

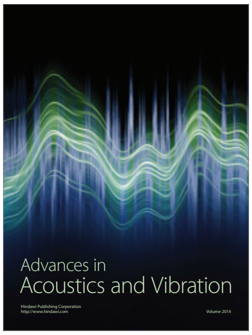
Journal of
Robotics


International Journal of
Chemical Engineering

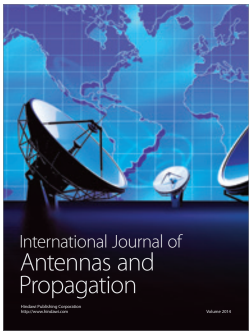
VLSI Design

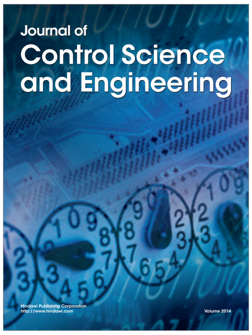
International Journal of
Navigation and Observation

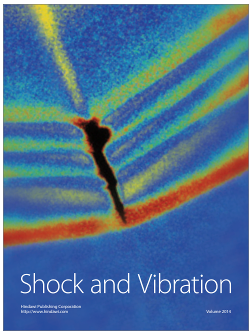

Modelling &
Simulation
in Engineering


Advances in
Acoustics and Vibration

Active and Passive
Electronic Components


International Journal of
Antennas and Propagation


Journal of
Control Science and Engineering


Shock and Vibration

Journal of
Electrical and Computer Engineering