SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2016; **9**:3452–3466
Published online 7 July 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1553

RESEARCH ARTICLE

Message-locked proof of ownership and retrievability with remote repairing in cloud

Jing Chen[1], Lihong Zhang[1], Kun He[1], Min Chen[2], Ruiying Du[1] and Lina Wang[1]

[1] State Key Laboratory of Software Engineering, Computer School, Wuhan University, Wuhan 430072, China
[2] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

ABSTRACT

Cloud storage services are widely deployed and employed in recent years. A number of data checking techniques have been proposed for secure cloud storage services. These state-of-the-art schemes only focus on some aspects, such as data integrity, users' ownership, and data resiliency, but the overall safety of cloud storage services is not discussed sufficiently. Considering cloud storage requirements as a whole, in this paper, we propose a model of message-locked proof of ownership and retrievability with remote repairing, which provides data confidentiality, secure cross-user deduplication at the client-side, file retrievability, ownership privacy-preserving, random block accessing, and remote repairing simultaneously. In addition, we also propose a concrete construction and prove its security in the random oracle model. The experimental results show that our construction is efficient in practice.

KEYWORDS
cloud storage; encryption; proof of ownership; proof of retrievability; remote repairing

***Correspondence**
Jing Chen, Computer School, Wuhan University, Wuhan 430072, China.
E-mail: chenjing@whu.edu.cn

1. INTRODUCTION

Cloud storage services (e.g., Amazon S3, Google Drive, and Microsoft Azure) have become one of the most popular applications in recent years [1–4]. With cloud storage services, users can escrow their files to the cloud storage servers and enjoy the on-demand high-quality applications and services, without the burden of local data storage and maintenance. Users can access their data anywhere anytime and even share them with others [5]. Both individuals and companies can obtain tremendous benefits from this kind of outsourcing services, such as mitigation of the responsibility to storage management, flexible accessibility with location independence, and avoidance of huge economic costs on hardware, software, and personnel maintenances [6]. Although cloud storage technology brings a number of advantages, it also causes many new security challenges that can be divided into two categories: client-side and server-side.

In one side, from the perspective of users, the first consideration is the authenticity of cloud storage service provider. Because of the risks of intrusion, failure, and privacy leakage, a practical cloud storage system should provide both data confidentiality and integrity for users. Encryption is supposed to the most common method to ensure data confidentiality [7]. As a result, in order to protect data privacy, users prefer encrypting their files before outsourcing. Message authentication codes and digital signatures are used to ensure data integrity in many systems. But these methods require the original files to accomplish verification, which means that users need to download the complete files before verifying. Ateniese, *et al*. [8] introduced provable data possession (PDP) to ensure data integrity. Employing a PDP scheme, users can verify the integrity of remote files without downloading them from the server. However, PDP only provides a probabilistic guarantee of data integrity. For example, the server can reverse one bit of each file and pass verification with overwhelming probability.

The second consideration of users is that a practical cloud storage system should provide file retrievability rather than isolated probabilistic guarantee of data integrity. To achieve file retrievability, Juels and Kaliski Jr. [9] introduced proof of retrievability (PoR). In a PoR scheme, users not only can verify the integrity of remote files but also can retrieve the original files if the verification succeeds. However, the repairing cost of error-correcting codes is expensive. Dimakis, *et al*. [10] proposed

regenerating code to reduce the repairing cost, but it only supports file-level accessing rather than random block accessing. In fact, generally, people only need to modify parts of the file blocks. Thus, a block-level coding scheme is a better choice.

In the other side, from the perspective of cloud storage service provider, the efficiency of cloud storage system is also a key factor to reduce costs and enhance users' friendliness, under the premise of security. File-level deduplication is an effective technique to reduce data redundancy at the server-side. That means, for a certain file, the server only maintains a single copy of the file to make efficient use of storage space and communication bandwidth, regardless of how many users ask to store that file. Therefore, if the file already exists on the server, users do not need to upload it again to the server. However, a malicious user may attempt to access a file of other users by convincing the server that it holds this file. To prevent such attacks, Halevi, *et al*. [11] introduced proof of ownership (POW) to realize secure deduplication at the client-side. As we mentioned that users prefer encrypting their files at the client-side, it is challenging to achieve cross-user deduplication, because identical files may be encrypted to different ciphertexts. Fortunately, Bellare, *et al*. [12] introduced a method called message-locked encryption to achieve secure deduplication for ciphertexts, which can offer tag consistency security and meet the strong privacy notion PRV$-CDA ("cda" stands for "chosen-distribution attack") defined in [12].

With aforementioned discussion, one may design the following system as a practical solution.

Strawman. If a client intends to upload a file, he or she first encrypts the file by encryption algorithm in [12] and encodes the encrypted file by error-correcting code as [9] does. Then he or she computes the hash value of the encoded file and sends it to the server. If the hash value matches some file stored on the server, the client and the server run the challenge–response protocol in [11] to achieve secure cross-user deduplication at the client-side. Otherwise, the client employs the publicly-verifiable variant of tagging algorithm in [8] and uploads the encoded file along with all tags to the server. Then, anyone who knows the public key can run the challenge–response protocol in [8] to verify the integrity.

Unfortunately, there are three major disadvantages in this strawman solution. The first one is computationally intensive. Almost all of the publicly-verifiable PDP or PoR schemes employ public-key cryptography technique, so the heavy computation is hardly avoided. And it is unnecessary to encode the file with error-correcting code if the client only needs to accomplish deduplication at the client-side (without uploading it to the server). The second one is privacy leakage. Because other clients need to use a public key to verify integrity, they can conclude the user who uploaded the file. The last drawback is the expensive cost for repairing damaged blocks as we mentioned. To repair a damaged block, a solution-based error-correcting code has to first retrieve the entire original file as explained in [10].

Given the concerns mentioned previously, we design a comprehensive model for such cloud storage systems. For clarity, we summarize the desired requirements here:

- **Data confidentiality.** The content of users' files should not be leaked via a malicious administrator or a server intruder.
- **Secure cross-user deduplication.** A user does not need to upload a file if the server stores the same one. In addition, a malicious user cannot convince the server that it holds a certain file just based on some short information about the file.
- **File retrievability.** Users can check the integrity of their files without downloading them, and if the server passes the verification, they can always recover the original files.
- **Ownership privacy-preserving.** No user should be able to learn the ownership of a file except the server and the specified users.
- **Random block accessing.** Users can access random blocks of a file without downloading or decoding the entire file.
- **Remote repairing.** When users discover that some blocks of a file are damaged, the server can repair them at the server-side.

Our contributions. The main contributions of our paper are summarized as follows:

(1) We design the model of *message-locked proof of ownership and retrievability* (or message-locked PoOR for short) and give the formal security definitions.
(2) We propose the first efficient construction of message-locked PoOR with remote repairing and prove its security under random oracle model.
(3) We implement our construction and compare it with other schemes. The experimental results show that our construction is efficient in practice.

Organization. The rest of this paper is organized as follows. We briefly discuss the related work in the next section. Then, we present the system model, threat model, and design goals in Section 3. In Section 4, we propose the model of message-locked PoOR, which is followed by a concrete scheme called (3R-PoOR) in Section 5. The security analysis and performance evaluation of our scheme are presented in Sections 6 and 7, respectively. In the last section, we conclude this paper.

2. RELATED WORK

Currently, many researchers concern the issue of secure cloud storage. This section reviews the related works from three aspects: data integrity, data retrievability, and secure cross-user deduplication, where the first two aspects are the secure requirements of users and the last one is the

requirement of cloud server. Especially, some schemes, belonging to these three types, also may partially satisfy some other requirements that we listed in Section 1.

To provide integrity protection for remote data, Ateniese, *et al*. [8] initially constructed PDP. They proposed two provably secure PDP schemes (S-PDP and E-PDP) based on homomorphic verifiable tags that support unlimited verification. But their schemes are computationally intensive. Based on [8], Ateniese, *et al*. [13] presented two provably secure PDP schemes that are more efficient than previous solutions. In follow-up work, improved studies [14–17] enhanced the capability of PDP from various aspects. However, a serious problem of PDP is that a corrupted sever, which loses a small portion of users' files, may still pass the verification. In that case, users cannot recover the original files.

On the other hand, to protect the integrity and recoverability of data stored on the cloud, Juels and Kaliski [9] introduced a closely related concept called PoR. In [9], the authors proposed an efficient sentinel-based scheme utilizing error-correcting code, but it only supports a limited number of queries. Several recent studies proposed some schemes that improve security and efficiency of PoR [18–21], and some other schemes that enhance the capability of PoR [22–24]. All of these researches employed existing coding algorithms that do not support efficient remote repairing. Dimakis, *et al*. [10] found that the repairing cost of error-correcting code was expensive in distributed storage systems. Then, they proposed a regenerating code to reduce the repairing cost. Their scheme does not support random block accessing, thus can only be used in read-rarely systems.

Moreover, to satisfy the requirement of secure cross-user deduplication, Halevi, *et al*. [11] introduced a solution called POW. The authors proposed three practical schemes based on Merkle tree that only a few hash values should be transmitted when the user convinces the server that it has a certain file. Pietro, *et al*. [25] enhanced [11] and proposed a secure POW scheme that reduces the computational cost to a constant number of pseudorandom function operations. Both of their solutions deal with the files in plaintext, and it is more challenging when the files are encrypted. Bellare, *et al*. [12] introduced message-locked encryption to solve deduplication for ciphertext, which is a generalization of convergent encryption. Unfortunately, their research does not support both file retrievability and ownership privacy-preserving. Zheng and Xu [26] proposed a solution that achieves POW and proof of storage, but their scheme cannot provide data confidentiality and ownership privacy-preserving.

Most of existing works just provide a partial solution of secure cloud storage; thus, a comprehensive solution, which covers data confidentiality, file retrievability, secure cross-user deduplication at the client-side, ownership privacy-preserving, random block accessing, and remote repairing simultaneously, becomes significant.

3. PROBLEM STATEMENT

In this section, we describe the system model and threat model of this paper at first. Then, we illustrate the design goals of our scheme.

3.1. System model

In this paper, the system model is composed of two main entities: the cloud server and users (as illustrated in Figure 1). The cloud server, which has large storage space and computation resource, offers data storage services to users. Users can be either individual consumers or organizations. The users upload their data (such as documents, pictures, and videos) to the cloud server and can access them anywhere anytime, which relieves the burden of users in storage and maintenance. However, the storage location of data and the concrete implementation of cloud storage services are transparent to users. We assume users' data are in form of files that are further divided into a number of blocks. To protect the integrity of the data, each block is attached with an authentication tag generated by users.

In the system, when a user $\mathcal{U}$ wants to outsource a file F, the cloud server firstly estimates whether F already

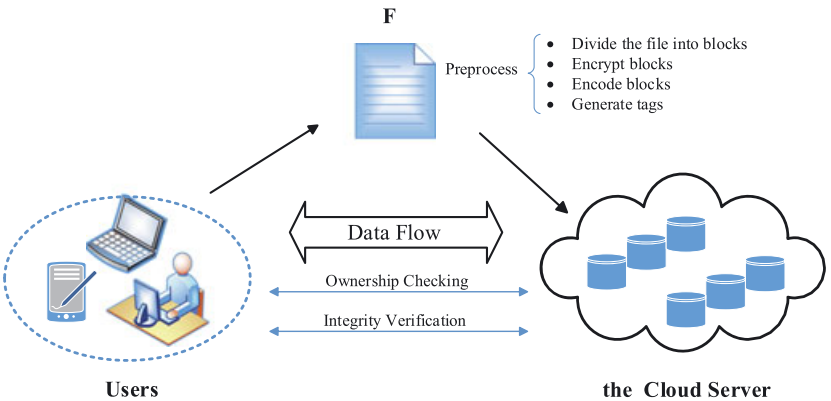


Figure 1. System model.

exists in cloud. If F has been stored in cloud and $\mathcal{U}$ passes the ownership checking, then $\mathcal{U}$ need not upload F and is marked as a valid owner of F by the cloud server. Otherwise, if F does not exist in cloud, $\mathcal{U}$ has to preprocess F and upload it. After the procedure of storing, $\mathcal{U}$ checks the integrity of F periodically to assure that F is intact in cloud.

The cloud server attempts to minimize the bandwidth and to make efficient use of its storage facility by deduplication at the client-side. As a result, before a user uploads a file, the cloud server should estimate whether the file has been uploaded by another user. In detail, at first, the user sends the hash value of the file to the cloud server, and then the server checks whether the hash value matches some existing hash value stored on the cloud. If so, the server generates a challenge to ask the user for the POW. Upon a successful challenge, the user need not to upload the file to the server again and is marked as an owner of the file by the server at the same time. On the contrary, if the file does not exist on the cloud, the user must encrypt and encode all blocks of the file. Additionally, each block has a unique authentication tag. After that, the user uploads the encoded file with all tags to the cloud server.

As users no longer possess their data locally, it is of critical importance for users to ensure that their data are being successfully stored and maintained. For checking the integrity of the files, users periodically challenge the cloud server by a randomly selected set of file blocks. With the queried blocks and their corresponding tags, the server generates a proof of data possession and sends it back to users. Then, users verify the proof by its local metadata. If the verification succeeds, users can retrieve the original files when they download these files. Otherwise, users can request the cloud server to exactly repair the damaged blocks if the number of damaged blocks is not greater than a threshold value.

3.2. Threat model

In our 3R-PoOR scheme, we consider four factors that may threaten the integrity of users' data stored on the cloud server. First, users' data stored on the cloud may be polluted inadvertently because of hardware/software failures and operational errors of system administrator. Second, the cloud service provider may modify and leak users' data illegally. Third, attackers may corrupt data on the cloud and prevent users from using data correctly. Last but not the least, in order to access a file uploaded by other users, a malicious user may cheat the server by claiming that it has this file, while it does not possess the whole file at all.

3.3. Design goals

To correctly and efficiently verify the integrity of users' data with message-locked POW and remote repairing, our scheme should achieve the following properties: (i) **Correctness**: Users are able to correctly check the integrity of their data without retrieving the entire data from the cloud. In particular, if the data are corrupted, the cloud server cannot forge valid integrity proof information. (ii) **Soundness**: Any malicious adversary without the whole file stored on the cloud cannot pass the ownership checking protocol. (iii) **Efficiency**: The cloud server can efficiently repair the damaged blocks without the help of the users.

4. MESSAGE-LOCKED POOR

In this section, we propose a novel model called message-locked PoOR and provide the formal security definitions. This model can protect the comprehensive security of cloud storage including data confidentiality, secure cross-user deduplication at the client-side, file retrievability, and ownership privacy-preserving simultaneously. Random block accessing and remote repairing depend on the underlying coding algorithm, which we discuss in the next section.

4.1. Syntax and correctness

Definition 1 (**Message-locked PoOR**). *A message-locked PoOR scheme consists of the following four algorithms and protocols:*

- $(id, e) \leftarrow Init(1^{\lambda}, F)$*, this deterministic initialization algorithm is run by a client, which takes as input the security parameter* λ *and the original file* F*, outputs the public identity* id *and the secret metadata* e*.*
- $(C, T) \leftarrow Encode(e, F)$*, this encoding algorithm is run by a client, which takes as input the metadata* e *and the original file* F*, outputs the encoded file* C *and the authenticator* T*.*
- $0/1 \leftarrow \mathrm{OCheck}\langle \mathcal{U}(e, F), \mathcal{S}(C, T)\rangle$*, this randomized ownership checking protocol is run by a client* $\mathcal{U}$ *and a server* $\mathcal{S}$ *interactively.* $\mathcal{U}$ *takes as input the metadata* e *and the original file* F*, while* $\mathcal{S}$ *takes as input the encoded file* C *and the authenticator* T*. This protocol outputs 1 if* $\mathcal{U}$ *convinces* $\mathcal{S}$ *that it possesses the complete file* F *locally and 0 otherwise.*
- $0/1 \leftarrow \mathrm{RCheck}\langle \mathcal{S}(C, T), \mathcal{U}(e)\rangle$*, this randomized retrievability checking protocol is run by a server* $\mathcal{S}$ *and a client* $\mathcal{U}$ *interactively.* $\mathcal{S}$ *takes as input the encoded file* C *and the authenticator* T*, while* $\mathcal{U}$ *takes as input the metadata* e*. This protocol outputs 1 if* $\mathcal{S}$ *convinces* $\mathcal{U}$ *that it faithfully stores* (C, T) *and 0 otherwise.*

Because the initialization algorithm is deterministic, two same files always have the same metadata and identity. The metadata serves as the secret key in a message-locked PoOR scheme, the size of which is much shorter than the size of file to avoid trivial constructions. However, unlike [12], the identity in our model is derived from the original file rather than the encoded file. It is because the identity in our model is used to decide whether the deduplication at the client-side should be conducted and the client

has not to fully encode the file if the file already exists on the server. The word "message-locked" means that the message is locked by itself, which is same as [12]. The message-locked encryption in our model can also achieve tag consistency and PRV$-CDA[12]. In other words, an adversary cannot make a honest client recover a file different from the one he or she uploaded, and the encryption of an unpredictable message is indistinguishable from a random string of the same length.

The encoding algorithm is the most time-consuming process in message-locked PoOR. Therefore, it is conducted only when necessary. The encoding algorithm can be probabilistic or deterministic. The output of this algorithm consists of two parts: the encoded file that is obtained from an encryption algorithm and a coding algorithm, and the authenticator that is obtained from a tagging algorithm.

In ownership checking protocol, the client needs to prove to the server that it possesses the whole original file locally, and this is why it takes the original file as input. When this protocol is executed, the client only needs to encode parts of the file on demand. However, the server can only take authenticator as input for efficiency consideration.

In retrievability checking protocol, the client only takes the metadata as input when it runs this protocol, because it deleted the file locally after uploading. The server needs to take the encoded file as input in this protocol; otherwise, the checking protocol can only verify the possession of the authenticator.

Definition 2 (**Correctness**). *A message-locked PoOR scheme is correct if the following conditions hold for any positive integer* λ*, any file* $F \in \{0,1\}^*$*, and any metadata* e *generated from Init*$(1^\lambda, F)$*:*

$$\Pr[\mathrm{OCheck}\langle \mathcal{U}(e,F), \mathcal{S}(Encode(e,F))\rangle = 1] \geq 1 - \epsilon_1(\lambda)$$
$$\Pr[\mathrm{RCheck}\langle \mathcal{S}(Encode(e,F)), \mathcal{U}(e)\rangle = 1] \geq 1 - \epsilon_2(\lambda)$$

where $\epsilon_1(\cdot)$ *and* $\epsilon_2(\cdot)$ *are two negligible functions.*

4.2. Security definition

The security of message-locked PoOR consists of four parts: indistinguishability, uncheatability, unforgeability, and nonidentifiability, which correspond to the requirements of data confidentiality, secure cross-user deduplication, file retrievability, and ownership privacy-preserving in Section 1. Only when message-locked PoOR satisfies indistinguishability, uncheatability, unforgeability, and nonidentifiability can the scheme achieve corresponding requirements in a secure way. The other two requirements random block accessing and remote repairing, which are not included in basic message-locked PoOR model and are not security requirements, will be discussed in the next subsection.

First of all, we think about indistinguishability that captures the requirement of data confidentiality. Our definition is similar to the privacy definition in [12]. But the adversary in our model can obtain message-locked authenticator. We start this definition from the distinguishing game between a challenger $\mathcal{C}$ who serves as the client and an adversary $\mathcal{A}$ who serves as the server:

(1) $\mathcal{C}$ chooses a bit $\beta \in \{0,1\}$ and a file vector $\mathbf{F}$ from the distribution given by $\mathcal{A}$. We require that the entries in $\mathbf{F}$ are pairwise distinct. For $j = 1, \ldots, |\mathbf{F}|$, $\mathcal{C}$ runs $(\mathbf{id}[j], \mathbf{e}[j]) \leftarrow \mathrm{Init}(1^\lambda, \mathbf{F}[j])$, $(\mathbf{C}_1[j], \mathbf{T}[j]) \leftarrow \mathrm{Encode}(\mathbf{e}[j], \mathbf{F}[j])$, $\mathbf{C}_0[j] \leftarrow \{0,1\}^{|\mathbf{C}_1[j]|}$ and sends $(\mathbf{id}, \mathbf{C}_\beta, \mathbf{T})$ to $\mathcal{A}$.
(2) $\mathcal{A}$ outputs β'. $\mathcal{A}$ wins if $\beta' = \beta$.

Definition 3 (**Indistinguishability**). *A message-locked PoOR scheme is indistinguishable if for any probabilistic polynomial time (PPT) adversary* $\mathcal{A}$ *and unpredictable source (see* [12] *for the details of unpredictable), the probability that* $\mathcal{A}$ *wins the distinguishing game is negligible greater than* 1/2*.*

Secondly, we consider the definition of uncheatability that captures the requirement of secure cross-user deduplication at the client-side. Intuitively, the client cannot cheat the server that it possesses the complete file except with negligible probability. As in the definition of indistinguishability, we require that the files have high min-entropy in the definition of uncheatability. We start this definition from the cheating game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:

(1) $\mathcal{C}$ chooses a file $F \leftarrow \{0,1\}^*$ from the distribution given by $\mathcal{A}$, runs $(id, e) \leftarrow \mathrm{Init}(1^\lambda, F)$, and sends (id, e) to $\mathcal{A}$.
(2) $\mathcal{A}$ queries the blocks of F for $q(\lambda)$ times, where $q(\cdot)$ denotes some polynomial.
(3) $\mathcal{C}$ and $\mathcal{A}$ run the ownership checking protocol. $\mathcal{A}$ wins if the protocol returns 1.

Definition 4 (**Uncheatability**). *A message-locked PoOR scheme is uncheatable if for any PPT adversary* $\mathcal{A}$ *and unpredictable source, the probability that* $\mathcal{A}$ *wins the cheating game is negligible.*

Thirdly, unforgeability captures the requirement of file retrievability. If the server passes the retrievability checking protocol, the client is able to retrieve the complete original file with overwhelming probability. We start this definition from the forging game between a challenger $\mathcal{C}$ who serves as the client and an adversary $\mathcal{A}$ who serves as the server:

(1) $\mathcal{C}$ chooses a file $F \leftarrow \{0,1\}^*$ from the distribution given by $\mathcal{A}$, runs $(id, e) \leftarrow \mathrm{Init}(1^\lambda, F)$, and sends (id, e) to $\mathcal{A}$.
(2) $\mathcal{A}$ that serves as the client and $\mathcal{C}$ that serves as the server run the retrievability checking protocol for $q_1(\lambda)$ times, where $q_1(\cdot)$ denotes some polynomial.

(3) $\mathcal{A}$ that serves as the server and $\mathcal{C}$ that serves as the client run the ownership checking protocol for $q_2(\lambda)$ times, where $q_2(\cdot)$ denotes some polynomial.
(4) $\mathcal{C}$ and $\mathcal{A}$ run the retrievability checking protocol that $\mathcal{C}$ serves as the client. $\mathcal{A}$ wins if the protocol returns 1.

Definition 5 (Unforgeability). *A message-locked PoOR scheme is unforgeable if for any PPT adversary $\mathcal{A}$ who wins the forging game, there exists an extractor who can recover F from $\mathcal{A}$ except with negligible probability.*

Finally, nonidentifiability captures the requirement of ownership privacy-preserving. Similarly, we give the definition by the identifying game between two challengers $\mathcal{C}_0$ and $\mathcal{C}_1$ who serve as the clients and an adversary $\mathcal{A}$ who serves as the server:

(1) $\mathcal{A}$ chooses a file $F \leftarrow \{0,1\}^*$ and gives it to $\mathcal{C}_0$ and $\mathcal{C}_1$. For $j = 0, 1$, $\mathcal{C}_j$ runs $(id_j, e_j) \leftarrow \text{Init}(1^\lambda, F)$, $(C_j, T_j) \leftarrow \text{Encode}(e_j, F)$ and gives (id_j, C_j, T_j) to $\mathcal{A}$. Then $\mathcal{A}$ runs $\text{OCheck}\langle \mathcal{C}_j(e_j, F), \mathcal{A}(C_j, T_j)\rangle$ and $\text{RCheck}\langle \mathcal{A}(C_j, T_j), \mathcal{C}_j(e_j)\rangle$ for one time.
(2) The system chooses a bit $\beta \in \{0,1\}$. $\mathcal{C}_\beta$ runs $(id, e) \leftarrow \text{Init}(1^\lambda, F)$, $(C, T) \leftarrow \text{Encode}(e, F)$ and gives (id, C, T) to $\mathcal{A}$. Then $\mathcal{A}$ runs $\text{OCheck}\langle \mathcal{C}_\beta(e, F), \mathcal{A}(C, T)\rangle$ and $\text{RCheck}\langle \mathcal{A}(C, T), \mathcal{C}_\beta(e)\rangle$ for one time.
(3) $\mathcal{A}$ outputs β'. $\mathcal{A}$ wins if $\beta' = \beta$.

Definition 6 (Nonidentifiability). *A message-locked PoOR scheme is perfect unidentifiable if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the identifying game is exactly 1/2.*

4.3. Extension

The original message-locked PoOR model does not require random block accessing and remote repairing, which are essential requirements in our environment. Therefore, we add the following two protocols in a message-locked PoOR scheme to capture these two requirements:

- $\perp/m_i \leftarrow \text{Read}\langle \mathcal{U}(i), \mathcal{S}(C)\rangle$, this deterministic accessing protocol is run by a client $\mathcal{U}$ and a server $\mathcal{S}$. $\mathcal{U}$ takes as input the index i; $\mathcal{S}$ takes as input the encoded file C. This protocol outputs the ith original block m_i if accessing is valid and $\perp$ otherwise. The scheme supports random block access if the block accessing complexity at the server-side is $O(1)$.
- $0/1 \leftarrow \text{Repair}\langle \mathcal{U}(e, i), \mathcal{S}(C)\rangle$, this deterministic repairing protocol is run by a client $\mathcal{U}$ and a server $\mathcal{S}$. $\mathcal{U}$ takes as input the metadata e and the index i; $\mathcal{S}$ takes as input the encoded file C whose ith block was damaged. This protocol outputs the 1 if repairing is successful and 0 otherwise. The scheme supports remote repairing if the repairing can be performed at the server-side without the help of clients.

These two extensions are useful for a practical cloud storage system. The main advantage of random block accessing is that the clients can access any block with minimum communication cost and computational cost. Furthermore, most of the existing solutions [9,27], and [23] only take the file retrievability into account. In this case, although the encoded file is damaged and the client does not find it, the original file can be recovered at the client-side. But what if the clients find the damage and the damaged blocks can be repaired? Those solutions require the clients download enough blocks to accomplish repairing, which can cause huge communication cost and computational cost at the client-side. However, remote repairing can reduce the cost of the clients. With remote repairing, the server can repair the damaged block without the help of the clients.

5. THE CONSTRUCTION OF 3R-POOR

In this section, we describe a concrete scheme called 3R-PoOR. It consists of six algorithms and protocols as illustrated in Section 4.

5.1. Building blocks

The main tools employed in our scheme are described next (we do not emphasize the length of inputs and outputs for simplicity):

- **Collision-resistant hash functions**: A hash function $\text{H} : \{0,1\}^* \rightarrow \{0,1\}^*$ is collision-resistant if it is impossible to find two different values x and y that satisfy $\text{H}(x) = \text{H}(y)$ except with negligible probability.
- **Keyed-hash message authentication codes**: A keyed-hash message authentication code $\text{HMAC}_k(x) : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is a deterministic function that takes a key k and an input x and outputs a value y.
- **Pseudorandom functions**: A pseudorandom function $f_k(x) : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is a deterministic function that takes a key k and an input x and outputs a value y that is indistinguishable from a true random function of the same input x.
- **Pseudorandom permutations**: A pseudorandom permutation $\pi_k(x) : \{0,1\}^* \times [1,l] \rightarrow [1,l]$ is a deterministic function that takes a key k and an input x where $1 \leq x \leq l$ and outputs a value y where $1 \leq y \leq l$ that is indistinguishable from a true random permutation of the same input x.
- **Key derivation functions**: A key derivation function $\text{KDF} : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is a deterministic function that can derive a secret key from some secret values.
- **Deterministic symmetric encryption functions**: A deterministic symmetric encryption function

$\mathrm{Enc}_k(m) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is a deterministic function that takes a key k and a plaintext m and outputs a ciphertext c.

5.2. The coding algorithm

Definition 5 requires that a client should be able to retrieve the complete original file if the server passes the retrievability checking protocol. But as mentioned in Section 1, unlike traditional message authentication codes and digital signatures, the retrievability checking protocol only provides a probabilistic guarantee of data integrity, for example, if a file has 10 000 blocks and the server deletes only one of them. Then, when a client challenges 460 blocks (this number is used in many of PDP and PoR schemes), the probability that the server can pass the verification is greater than 95%. To achieve file retrievability, we have to employ data redundancy technique [9,23,28,29], such as error-correcting codes and erasure codes.

Roughly speaking, a $(n+\gamma, n, d)$ erasure code expands n blocks to $n+\gamma$ blocks and can tolerate at most $d-1$ damaged blocks. Combining retrievability checking protocol and erasure code can achieve the requirement of file retrievability. That is, if the server manipulated more than $d-1$ blocks, the client can discover the misbehavior with overwhelming probability; otherwise, the client can retrieve the original file with the help of the erasure code. When the client discovers that some blocks are damaged, it is possible to repair them if the number of total damaged blocks is less than d. But it needs to retrieve the complete file before repairing the damaged blocks, which causes heavy computational cost and communication cost. If the server knows the eraser coding algorithm, the repairing work can be performed at the server-side, but the input/output cost and the communication cost are high in the distributed servers environment. Regenerating code in [10] can reduce communication cost in repairing and does not need to retrieve the complete file, but it does not support random block accessing that is an essential requirement in our model.

Given the aforementioned concerns, we employ a linear coding algorithm that satisfies our requirements. We describe the coding algorithm in single server environment, and the situation of distributed servers environment is beyond the scope of this paper. To achieve random block accessing, we have to use a systematic code; therefore, the output of the coding algorithm contains the input. To achieve remote repairing, the server needs to know all of the coefficients and corresponding coded blocks.

Assuming a file F consists of n original blocks, the server allows the clients to add γ coded blocks that satisfies $\gamma < n$, and then the expansion factor is $(n+\gamma)/n$. The clients prefer that the threshold value is δ. If the server manipulates more than δ blocks, the clients can discover the misbehavior except with negligible probability; otherwise, they can retrieve the original file with overwhelming probability. Because we need to retrieve n original blocks, δ should not be greater than γ. Then we have $0 < \delta \leq \gamma < n$. If $\delta = \gamma$, we call it maximum distance separable code.

Unlike erasure codes that focus on the storage-tolerance trade-off and regenerating codes that focus on the storage-bandwidth trade-off, we focus on the storage-computation trade-off. To capture this trade-off, we introduce a parameter β that denotes the minimum number of original blocks in a coded block. Because we need to ensure that a malicious server cannot delete all of the blocks that contain a certain original block without been discovered, each original block should be encoded into at least δ coded blocks. Then, we have $\delta n/\gamma \leq \beta \leq n$. If a code is maximum distance separable, we can obtain $\beta = n$, which indicates that the server has to access n blocks to repair one damaged block. Therefore, we ask for a better repairing performance. That is, we cannot employ a maximum distance separable code, and for a fixed ratio $\delta/(n+\gamma)$, we should balance γ and β.

For a given $(n, \gamma, \delta, \beta)$, a coded block is the linear combination of β successive original blocks. We can choose one seed for each coded block to generate β random coefficients in some finite field. As shown in [30], if the finite field is sufficiently large, the server can repair the original blocks with overwhelming probability.

5.3. The initialization algorithm and the encoding algorithm

Everyone who intends to upload a file should run the initialization algorithm, but only the client who actually uploads the file runs the encoding algorithm. The initialization algorithm takes as input the security parameter λ that decides the length of output and the original file F, computes the metadata e and the identity id as follows:

$$e = \mathrm{H}(F),\ id = \mathrm{H}(e)$$

The identity id is used to decide whether the deduplication at the client-side should be conducted as explained in Section 4.

The encoding algorithm is shown in Figure 2. The key for encrypting and tagging is randomly chosen from a key

$(C, T) \leftarrow \mathrm{Encode}(e, F)$:

1. Parse F as $(m_1, \ldots, m_n)$.
2. Choose $k \leftarrow \{0,1\}^{|e|}$, compute $r = k \oplus e$.
3. For $1 \leq j \leq n$, compute $k_j = \mathrm{KDF}(k, j)$, $c_j = \mathrm{Enc}_{k_j}(m_j)$.
4. Choose $s \leftarrow \{0,1\}^{|e|}$, for $1 \leq j \leq \gamma$, compute the seed $s_j = f_s(j)$ and c_{n+j}, where c_{n+j} is the linear combination of β encrypted blocks.
5. Compute $k_p = \mathrm{KDF}(k, 0)$, for $1 \leq j \leq n$, compute $t_j = f_{k_p}(j) + f_{k_p}(0)c_j$, and for $n+1 \leq j \leq n+\gamma$, compute $t_j = f_{k_p}(j \| s_{j-n}) + f_{k_p}(0)c_j$;
6. Set $C = (r, s, \{c_j\})$,$T = (r, s, \{t_j\})$.

Figure 2. The encoding algorithm.

space rather than the metadata itself, which makes our encoding algorithm probabilistic. The encryption method is similar to counter mode; thus, the client can encrypt parts of the file on demand. *s* is used to generate the seed of random coefficients as explained in Section 5.2. We employ homomorphic authenticator as [20] does, but our authenticator is slightly different from theirs. They use a random secret value to construct the tag, while we use a value that is interrelated with the file. In this way, each client that possesses the file can generate the same tag. Notice that, the tagging algorithm for the encoded blocks is slightly different from the algorithm for the original blocks, which is because we to bind the seed and the encoded blocks together.

5.4. The ownership checking protocol

If the client claims it has a certain file that exists on the server, the server runs the ownership checking protocol to verify the ownership. The detail of this protocol is shown in Figure 3. Both the client and the server employ the pseudorandom permutation to generate a subset of indexes that indicates *b* challenged blocks. Notice that the client only needs to prove that it has the original blocks, therefore *b* challenged blocks are chosen from *n* original blocks. Because the coding algorithm in our construction is systematic code and the encryption mode is similar to counter, the client only needs to encrypt parts of the file on demand and does not need to compute any encoded blocks. This reduces much computational cost in the ownership checking protocol. The server verifies v without the help of the tags, which means that the tags only serve for the retrievability checking protocol in our scheme.

5.5. The retrievability checking protocol

At any time, the client who has the ownership of the file can run the retrievability checking protocol to determine the integrity of the file. The detail of this protocol is shown in Figure 4. Unlike the ownership checking protocol, client has to verify all of the blocks stored on the server in the retrievability checking protocol, including original blocks and encoded blocks. Therefore *b* challenged blocks should be chosen from $n + \gamma$ blocks. We compress the proof by using homomorphic authenticators as [8] and [20] do. Note that this protocol only provides a probabilistic guarantee that the server did not manipulate or delete the blocks. The security of file retrievability is achieved by combining this protocol and the underlying coding algorithm.

$0/1 \leftarrow \mathrm{OCheck}\langle \mathcal{U}(e,F), \mathcal{S}(C,T)\rangle$:

1. $\mathcal{S}$ chooses $b \leftarrow [1,n]$, $r_0, r_1 \leftarrow \{0,1\}^\lambda$ and sends (b, r_0, r_1, r) to $\mathcal{U}$.
2. $\mathcal{U}$ executes following instructions:
   (a) Computes $k = r \oplus e$.
   (b) For $1 \le j \le b$, computes $i_j = \pi_{r_0}(j)$, $\kappa_j = \mathrm{KDF}(r_1, j)$, $k_{i_j} = \mathrm{KDF}(k, i_j)$.
   (c) Computes $v = \bigoplus_{j=1}^{b} \mathrm{HMAC}_{\kappa_j}(\mathrm{Enc}_{k_{i_j}}(m_{i_j}))$ and sends v to $\mathcal{S}$.
3. $\mathcal{S}$ executes following instructions:
   (a) For $1 \le j \le b$, computes $i_j = \pi_{r_0}(j)$, $\kappa_j = \mathrm{KDF}(r_1, j)$.
   (b) Returns 1 if $\bigoplus_{j=1}^{b} \mathrm{HMAC}_{\kappa_j}(c_{i_j})$ is equal to v and 0 otherwise.

Figure 3. The ownership checking protocol.

$0/1 \leftarrow \mathrm{RCheck}\langle \mathcal{S}(C,T), \mathcal{U}(e)\rangle$:

1. $\mathcal{U}$ chooses $b \leftarrow [1, n+\gamma]$, $r_0, r_1 \leftarrow \{0,1\}^\lambda$ and sends (b, r_0, r_1) to $\mathcal{S}$.
2. $\mathcal{S}$ executes following instructions:
   (a) For $1 \le j \le b$, computes $i_j = \pi_{r_0}(j)$, $a_j = f_{r_1}(j)$.
   (b) Computes $t = \sum_{j=1}^{b} a_j t_{i_j}$, $\rho = \sum_{j=1}^{b} a_j c_{i_j}$ and sends (t, ρ, r, s) to $\mathcal{V}$.
3. $\mathcal{U}$ executes following instructions:
   (a) Computes $k = r \oplus e$, $k_p = \mathrm{KDF}(k, 0)$.
   (b) For $1 \le j \le b$, computes $i_j = \pi_{r_0}(j)$, $a_j = f_{r_1}(j)$.
   (c) For $1 \le j \le b$, computes $\sigma_1 = \sum a_j f_{k_p}(i_j)$ if $1 \le i_j \le n$ and $\sigma_2 = \sum a_j f_{k_p}(i_j \| f_s(j-n))$ if $n+1 \le i_j \le n+\gamma$.
   (d) Returns 1 if $\sigma_1 + \sigma_2 + f_{k_p}(0)\rho$ is equal to t and 0 otherwise.

Figure 4. The retrievability checking protocol.

5.6. The accessing protocol and the repairing protocol

The accessing protocol is straightforward. When the client intends to access the *i*th original block, the server simply returns (r, c_i) to the client if $1 \le i \le n$. The client obtains m_i if c_i is successfully decrypted and $\perp$ otherwise. In this way, clients can access any block of original file with minimum communication cost and computational cost.

When the retrievability checking protocol fails, the client runs the repairing protocol to exactly repair the damaged blocks if the number of damaged blocks is not greater than δ. The detail of this protocol is shown in Figure 5. Supposing the *i*th block is identified a faulty block, we should consider whether its tag is correct. Specially, if the tag of the faulty block is damaged, the client needs to generate the tag again after the faulty block is successfully repaired. As long as all of the materials used in repairing are valid and correct, the server can repair the damaged

$0/1 \leftarrow \text{Repair}\langle \mathcal{U}(e,i), \mathcal{S}(C)\rangle$:

1. If the tag of the damaged block exists:
   (a) $\mathcal{U}$ sends the index of the damaged block i to the $\mathcal{S}$, $1 \le i \le n+\gamma$.
   (b) $\mathcal{S}$ computes $\rho = \sum_{j=1}^{\beta} a_{\alpha_j} c_{i_{\alpha_j}}$.
   (c) $\mathcal{U}$ runs RCheck to check the integrity of the block c_i.
   (d) $\mathcal{U}$ returns 1 If the result of RCheck is 1, and 0 otherwise.
2. Otherwise:
   (a) $\mathcal{U}$ sends the index of the damaged block i to the $\mathcal{S}$, $1 \le i \le n+\gamma$.
   (b) $\mathcal{S}$ executes following instructions:
      i. Computes $t = \sum_{j=1}^{\beta} a_{\alpha_j} t_{i_{\alpha_j}}, \rho = \sum_{j=1}^{\beta} a_{\alpha_j} c_{i_{\alpha_j}}$
      ii. Sends $(i_{\alpha_1}, \ldots, i_{\alpha_\beta}, a_{\alpha_1}, \ldots, a_{\alpha_\beta}, t, \rho, r, s)$ to $\mathcal{U}$
   (c) $\mathcal{U}$ executes following instructions:
      i. Runs RCheck to check whether the β blocks are correct.
      ii. Return 1 If the result of RCheck is 1, $c_i = \rho$; and 0 otherwise.
      iii. Computes the tag of the damaged block t_i if the damaged block is successfully repaired, and sends t_i to $\mathcal{S}$.

Figure 5. The repairing protocol.

block successfully without the help of the client, which can efficiently reduce the repair cost of the clients. For a well-designed coding algorithm, the indexes $i_{\alpha_1}, \ldots, i_{\alpha_\beta}$ and the coefficients $a_{\alpha_1}, \ldots, a_{\alpha_\beta}$ need not to be sent to the client; thus, the communication cost can be further reduced.

6. SECURITY ANALYSIS

6.1. Security proof

In this subsection, we prove the security of 3R-PoOR. All theorems are proved under the random oracle model. For each proof, we only give the proof sketch because of the page limited.

Theorem 1. *Let* H *be a random oracle, 3R-PoOR is indistinguishable if the deterministic symmetric encryption is both KR-secure and ROR-secure and the pseudorandom function is secure.*

Proof. (sketch) This proof consists of a series of games. The first game is exactly the same with the distinguish game. In the second game, the hash values are chosen from a uniform distribution, which indicates the hash function is viewed as the random oracle. Thus, the probability that the adversary can distinguish the first game and the second game is negligible. In the third game, the tags are chosen from a uniform distribution. Because the pseudorandom function is secure, the probability that the adversary can distinguish the second game and the third game is negligible. This reduces our scheme to remote command execution in [12]. That is, our scheme is secure if remote command execution is secure. □

Theorem 2. *Let* H *be a random oracle, 3R-PoOR is uncheatable if the keyed-hash message authentication code is secure.*

Proof. (sketch) Because the keyed-hash message authentication code is secure, the adversary cannot forge a valid output without the help of the valid input. In Definition 4, the files have high min-entropy; thus, even if the adversary queries parts of the file, the challenged blocks still contain unknown block with high probability. □

Obviously, the security guarantee of 3R-PoOR on uncheatability is weaker than the schemes in [11]. To achieve higher security, we can employ a Merkle tree as [11] does that also leads to a higher cost.

Theorem 3. *3R-PoOR is unforgeable if the pseudorandom function is secure.*

Proof. (sketch) This proof consists of two parts. The first part proves that the adversary cannot generate a valid proof except it uses the correct blocks and tags, and an extractor can extract the challenged blocks from the valid proof. The second part proves that there exist sufficient blocks to recover the original file if the verification succeeds.

To prove the first part, we employ a series of games. The first game is exactly the same with the forge game. In the second game, the hash values are chosen from a uniform distribution, which indicates the hash function is viewed as the random oracle. Thus, the probability that the adversary can distinguish the first game and the second game is negligible. In the third game, the tags are chosen from a uniform distribution. Because the pseudorandom function is secure, the probability that the adversary can distinguish the second game and the third game is negligible. In the last game, we can conclude the adversary who returns unexpected response. At last, we can prove that a valid proof is always obtained from the correct blocks and tags. Then an extractor can solve a system of linear equations to extract the challenged blocks. The extractor can gain sufficient blocks in polynomial time with non-negligible probability.

To prove the second part, we just need to examine the capability of code. The threshold value δ provides the guarantee that there are sufficient blocks with overwhelming probability if the verification succeeds. Because the

coefficients are sufficiently large, we can recover the original blocks with overwhelming probability. □

Theorem 4. *3R-PoOR is perfectly unidentifiable.*

Proof. (sketch) Because k in the encoding algorithm and r_0, r_1 in the ownership checking protocol and the retrievability checking protocol are chosen from a uniform distribution, every client can obtain the same output with the same probability. As a result, the adversary cannot determine who it is interacting with from the distribution of output. □

6.2. Detection probability analysis

Our 3R-PoOR scheme provides the probabilistic guarantees of files ownership and data integrity. Now, we analyze the detection probability in the ownership checking protocol and the retrievability checking protocol.

On one hand, we concern the probability that the server succeeds in the ownership checking protocol. On the other hand, we pay attention to the probability that the client succeeds in the retrievability checking protocol. However, because these two protocols both utilize "random sampling" to achieve misbehavior detection, the probabilistic guarantees in the two phases are identical.

Suppose the original file F is divided into n blocks, out of which x blocks are missed. Let b be the number of queried blocks in a challenge. Let X be a discrete random variable that denotes the number of missed blocks that have been detected. P_X is defined to represent the probability that at least one of the missed blocks is detected. So, we have

$$P_X = P\{X \geq 1\} = 1 - P\{X = 0\}$$
$$= 1 - \frac{C_{n-x}^{b}}{C_{n}^{b}} = 1 - \prod_{i=0}^{b-1} \frac{n-x-i}{n-i}$$

It follows that

$$1 - \left(1 - \frac{x}{n}\right)^{b} \leq P_X \leq 1 - \left(1 - \frac{x}{n-(b-1)}\right)^{b}$$

Generally, $b - 1 \ll n$, and then the two sides of the inequation are approximately equal. Therefore, we can have

$$P_X \approx 1 - \left(1 - \frac{x}{n}\right)^{b}$$

In this way, the approximate minimum b queried in a challenge can be expressed as

$$b = \lceil log_{1-\frac{x}{n}}(1 - P_X) \rceil$$

When x is a fraction of n, the misbehavior can be detected with a certain probability by querying a certain amount of blocks in a challenge, which is independent of the total number of file blocks. For example, to achieve the detection probability P_X = 99 %, 458, 90, 44, 28 blocks should be asked for proof in a challenge, respectively, for x = 1 %, 5%, 10%, 15% of n. Figure 6 plots b for different values of n and x, when the detection probability P_X is 99%.

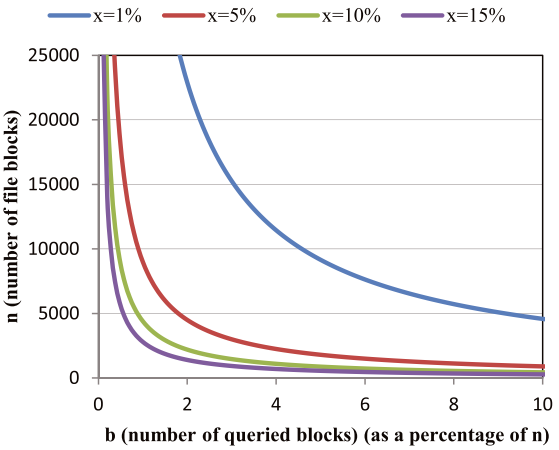


Figure 6. The number of queried blocks for different values of t, where $P_X = 99\%$.

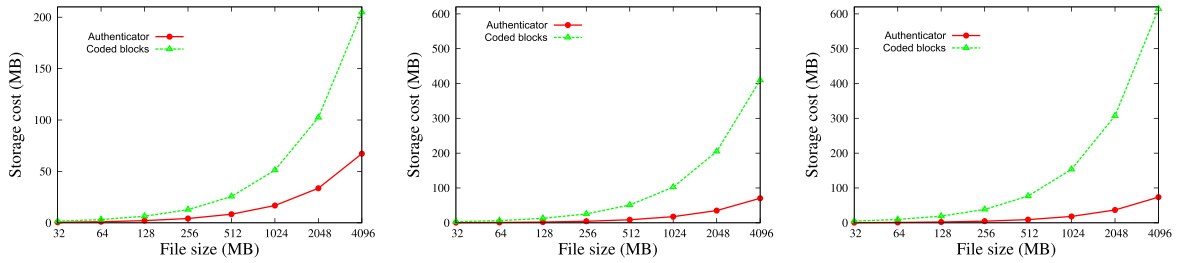
7. PERFORMANCE EVALUATION

7.1. Theoretical analysis

In this section, we carry out a theoretical analysis of 3R-PoOR scheme's performance and compare it with three other schemes: POW[11], proof of storage with deduplication (POSD)[26], and PDP[13]. For simplicity, in the rest of this paper, we use *Mul* and *Exp* to denote the complexity of one multiplication operation and one exponentiation operation on Group G, respectively. Let *H* and *XOR* denote the complexity of one hashing operation and one *XOR* operation, respectively. n is the number of blocks in an original file, s is the number of sectors in each block, and b is the number of blocks required in a challenge.

7.1.1. Storage cost analysis.

The extra storage cost at the server-side includes two parts: the coded blocks and the authenticator. The number of coded blocks depends on the size of γ, which is related to the fault-tolerant ability of scheme. Because the size of a tag is as large as a data block, the storage cost of tags is $O(n)$. To reduce the storage cost of the tags, we can divide each block into 64 sectors. Consequently, for a 4 KB block, the tag is only 64 bytes. And the authenticator is almost 18 MB for a 1 GB file. Figure 7 presents the server-side extra storage cost for different file sizes and different values of γ. As shown in Figure 7, the extra storage cost at the server-side is proportional to the file size and to the value of γ. When $\gamma = 0.10n$, the total extra storage cost at the server-side is 120 MB for a 1 GB file.



(a) $\gamma = 0.05n$ (b) $\gamma = 0.10n$ (c) $\gamma = 0.15n$

Figure 7. The extra storage cost at the server-side for different file size. The n is the number of blocks in the original file, and γ denotes the number of coded blocks.

Table I. Theoretical analysis and comparison of performance.

	3R-PoOR	POW[11]	POSD[26]	PDP[13]
Privacy preserving	yes	no	no	no
Ownership checking	yes	yes	yes	no
Retrievability checking	yes	no	yes	yes
Random block accessing	yes	yes	yes	yes
Remote repairing	yes	no	no	no
OCheck comp.cost(client)	$O(b)H + O(b)XOR$	$O(n^2)H$	$O(bs)Mul$	N/A
OCheck comp.cost(server)	$O(b)H + O(b)XOR$	$O(b \log n)H$	$O(bs)Mul + O(b)Exp$	N/A
OCheck comm.cost	$O(1)$	$O(b \log n)$	$O(s)$	N/A
RCheck comp.cost(server)	$O(b)Mul$	N/A	$O(bs)Mul + O(b)Exp$	$O(b)Mul + O(b)Exp$
RCheck comp.cost(client)	$O(b)Mul + O(1)XOR$	N/A	$O(bs)Mul + O(b)Exp$	$O(b)Mul + O(b)Exp$
RCheck comm.cost	$O(1)$	N/A	$O(s+b)$	$O(1)$

3R-PoOR, randomized proof of ownership and retrievability with remote repairing; POW, proof of ownership; POSD, proof of storage with deduplication; PDP, provable data possession.

7.1.2. Computing cost analysis.

In 3R-PoOR of our scheme, the initialization algorithm and the encoding algorithm are prepossessing procedures, which can be performed by the data owner off-line and will not influence the real-time performance. In the ownership checking protocol, to generate a proof of file ownership, the client has to perform bH and $bXOR$ operations. The server then verifies the accuracy of the ownership proof with $bH + (b-1)XOR$ operations. In the retrievability checking protocol, $2bMul$ operations need to be conducted by the server to produce the integrity proof. On receiving the proof, the client needs to perform $1XOR + (b+1)Mul$ operations to check the data integrity.

Now, we compare our 3R-PoOR scheme with PDP, POW, POSD and show the results in Table I. The client has to conduct $O(n^2)H$ operations to generate the proof in POW, and the computing cost of server to check the proof is $O(b \log n)H$, while our 3R-PoOR scheme just needs $O(b)H + O(b)XOR$ operations for both the client-side and the server-side in ownership checking phase. In POSD, to prove the proof, $O(bs)Mul + O(b)Exp$ operations need to be conducted for both ownership checking and retrievability checking, which makes the computing complexity high. Considering only the retrievability checking process, PDP requires $O(b)Mul + O(b)Exp$ computing cost at the client-side and server-side, which is higher than our 3R-PoOR scheme.

7.1.3. Communication cost analysis.

The communication cost in 3R-PoOR scheme is mainly caused by the challenging message and the proof information in the ownership checking protocol and the retrievability checking protocol. However, the challenging message is constant, the cost of which is $O(1)$ and can be ignored. In the ownership checking protocol, the client sends the ownership proof $\{v\}$ to the server, where v is a constant value generated by hash operations. Thus, the communication cost of this phase is $O(1)$. In the retrievability checking protocol, the size of an integrity proof $\{t, \rho, r, s\}$ is $|m|+|t|+2|p|$ bits, where $|m|$ is the size of a block, $|t|$ is the size of a tag, $|p|$ is the size of an element of G or Z_p. Therefore, the communication complexity of retrievability checking in our scheme is also $O(1)$.

We now compare our 3R-PoOR scheme with PDP, POW, POSD and summerize the results in Table I. In POW, to prove the file ownership, the client has to send the

hash values of sibling nodes for each queried block to the server, which incurs the communication cost as $O(b \log n)$. In POSD, the ownership proof includes s aggregated data blocks, so the communication overhead is $O(s)$. The communication cost of retrievability checking in POSD is $O(b + s)$. However, our 3R-PoOR scheme achieves $O(1)$ communication complexity for both ownership checking and retrievability checking. The communication cost of retrievability checking in PDP is also $O(1)$, but PDP cannot support ownership checking.

7.1.4. Repairing cost analysis.

At last, we analyze the cost in the repairing protocol. In our scheme, to repair one block, the server just needs to access β blocks and executes βMul operations. It is much smaller than the eraser code based solutions, in which the server needs to access n blocks. On the other hand, the process of repairing is achieved at the server-side in our scheme. Consequently, the server need not send the n blocks to the data owner and just proves the integrity of the β blocks used in remote repairing, which reduces the communication cost efficiently.

7.2. Experimental results

To show that our proposed 3R-PoOR scheme is efficient and scalable, we conducted experiments on a computer with Intel 2.8 GHz central processing unit and 4 GB memory using JAVA. As a basis for comparison, we have also implemented the schemes of Halevi, *et al.* [11] (POW), Zheng, *et al.* [26] (POSD), and Ateniese, *et al.* [13] (PDP). In our implementation, $\gamma = 0.10n$, which indicates the size of the encoded file, grows 10% with the original file. We set the size of each data block as 4 KB in our experiments. All experimental results represent the mean of 20 trials.

We estimate the running time in terms of two phases: ownership checking and retrievability checking. However, the input/output time to access the challenging blocks from disks is out of consideration in our experiments, which increases with the number of file blocks. Figure 8 indicates the computing time of ownership checking process for different file sizes in 3R-PoOR, POW, and POSD. To show the efficiency of our 3R-PoOR scheme, we change the size of files from 32 MB to 1 GB and keep the number of challenging blocks as 20. As shown in Figure 8, the computing time of POW is proportional to the size of files, while the computing time of 3R-PoOR and POSD both are constant. Compared with POSD, our 3R-PoOR scheme takes less time, which is consistent with our theory analysis. Figure 9 indicates the computational cost of client-side

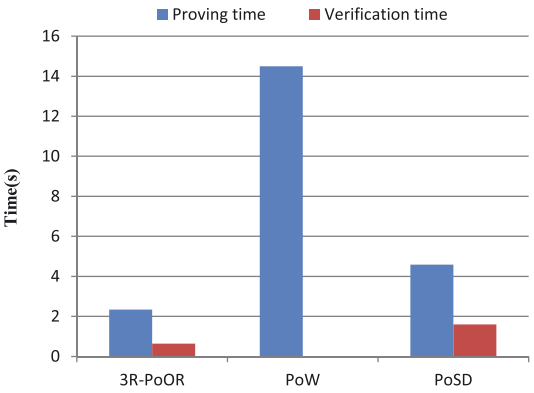


Figure 9. The computational cost of proving and verification in the ownership checking protocol. 3R-PoOR, randomized proof of ownership and retrievability with remote repairing; POW, proof of ownership; POSD, proof of storage with deduplication.

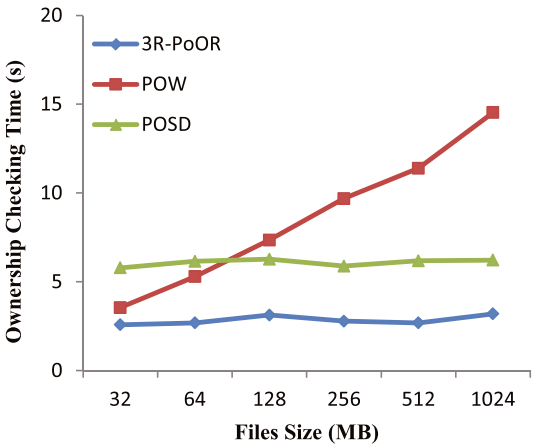


Figure 8. The computational cost comparison of ownership checking for different file sizes. 3R-PoOR, randomized proof of ownership and retrievability with remote repairing; POW, proof of ownership; POSD, proof of storage with deduplication.

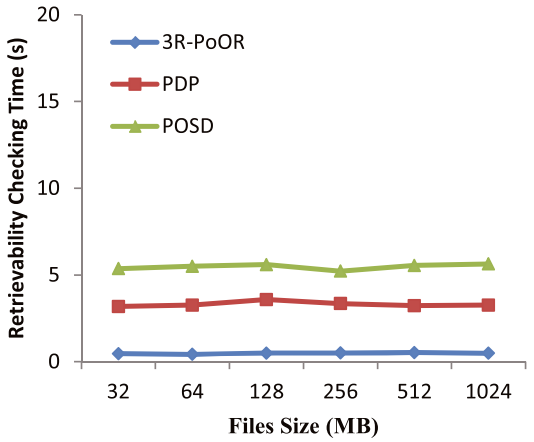


Figure 10. The computational cost comparison of retrievability checking for different file sizes. 3R-PoOR, randomized proof of ownership and retrievability with remote repairing; PDP, provable data possession; POSD, proof of storage with deduplication.

and server-side in the ownership checking protocol, where the size of file is 1 GB. The computing time of generating proof in POW and POSD is much higher than that in our 3R-PoOR scheme, because POW asks to construct the Merkle tree and POSD employs exponent calculation in this phase. The total computational cost of ownership checking in 3R-PoOR is only about 3 s for a 1 GB file.

Figure 10 depicts the computing time of retrievability checking process for different files in 3R-PoOR, PDP, and POSD . In order to achieve a higher detection probability, we set the number of challenging blocks in retrievability checking process as 480. From Figure 10, we can observe that the size of files has no influence on the performance of the three schemes when the number of challenging blocks is fixed. Among the three schemes, however, our 3R-PoOR spends the minimal time in the retrievability checking phase. Figure 11 compares the computational cost of 3R-PoOR, PDP, and POSD at client-side and server-side when the file size is 1 GB. The time spent to verify the proof in 3R-PoOR is 150 in our experiments, which is a little higher than PDP, and much less than POSD. Additionally, our 3R-PoOR scheme just spends around 330 ms to generate the integrity proof, which significantly outperforms PDP and POSD. To estimate the performance of 3R-PoOR with different challenging blocks, we vary the number of challenging blocks from 100 to 500, where the file size is 1 GB. As shown in Figure 12, when the size of file is fixed, the computing time increases with the number of challenging blocks in 3R-PoOR, PDP, and POSD. However, when the two parameters are same, our scheme takes less time than PDP and POSD.

Finally, we present the communication cost comparison for a 1 GB file in Table II, where the number of challenging blocks in OCheck and RCheck is 20 and 480, respectively. It can be shown that our 3R-PoOR scheme just needs to send 32 bytes in the process of ownership checking, which is the minimal communication overhead among these schemes. As POW has to send the sibling nodes for each path as discussed in Section 7.1, the communication cost in POW is the highest. For retrievability checking, the communication overhead of 3R-PoOR is higher than PDP, but PDP only supports integrity checking of storage data.

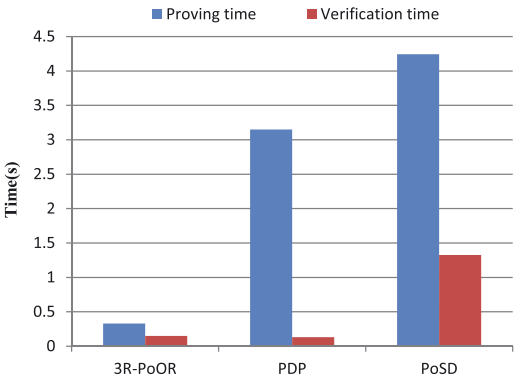


Figure 11. The computational cost of proving and verification in the retrievability checking protocol. 3R-PoOR, randomized proof of ownership and retrievability with remote repairing; PDP, provable data possession; POSD, proof of storage with deduplication.

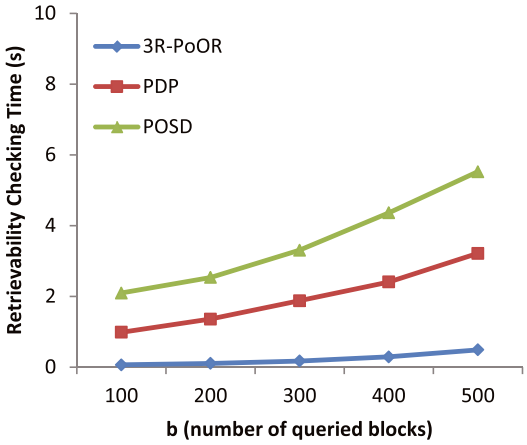


Figure 12. The computational cost comparison of retrievability checking for different queried blocks. 3R-PoOR, randomized proof of ownership and retrievability with remote repairing; PDP, provable data possession; POSD, proof of storage with deduplication.

Table II. The communication cost comparison for 1 GB file (byte).

	3R-PoOR	POW	POSD	PDP
OCheck comm.cost	32	8960	4096	–
RCheck comm.cost	4485	–	35 008	96

3R-PoOR, randomized proof of ownership and retrievability with remote repairing; POW, proof of ownership; POSD, proof of storage with deduplication; PDP, provable data possession.

8. CONCLUSION

In this paper, we introduce the comprehensive requirements in the cloud storage systems, which include data confidentiality, secure cross-user deduplication at the client-side, file retrievability, ownership privacy-preserving, random block accessing, and remote repairing. Based on these requirements, we propose the model of message-locked PoOR with remote repairing with formal security definitions. We also propose a construction called 3R-PoOR and prove its security under the random oracle model. To the best of our knowledge, it is the first construction that satisfies all the requirements simultaneously. The experimental results show that our construction is efficient in practice.

ACKNOWLEDGEMENTS

This research was supported in part by the Key Laboratory of Aerospace Information Security and Trusted

Computing, Ministry of Education, the National Natural Science Foundation of China under grant no. 61272451, 61572380, and U1536204, the Major State Basic Research Development Program of China under grant no. 2014CB340600, and the National High Technology Research and Development Program ("863" Program) of China under grant no. 2014BAH41B00.

REFERENCES

1. Armbrust M, Fox A, Griffith R, *et al.* A view of cloud computing. *Communications of the ACM* 2010; **53**(4): 50–58.
2. Wang Z, Liao J, Cao Q, Qi H, Wang Z. Friendbook: a semantic-based friend recommendation system for social networks. *IEEE Transactions on Mobile Computing* 2015; **14**(3): 538–551.
3. Chen M, Hao Y, Li Y, Lai CF, Wu D. On the computation offloading at ad hoc cloudlet: architecture and service models. *IEEE Communications* 2015; **53** (6): 18–24.
4. Chen M, Qian Y, Mao S, Tang W, Yang X. Software-defined mobile networks security. *Mobile Networks and Applications* 2016: 1–15. DOI: 10.1007/s11036-015-0665-5.
5. He K, Chen J, Du R, Wu Q, Xue G, Zhang X. Deypos: deduplicatable dynamic proof of storage for multi-user environments. *IEEE Transactions on Computers* 2016, DOI: 10.1109/TC.2016.2560812.
6. Velte T, Velte A, Elsenpeter R. *Cloud Computing, a Practical Approach*, 1st edn. McGraw-Hill, Inc.: New York, NY, USA, 2010.
7. Xia Z, Wang X, Sun X, Wang Q. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems* 2016; **27**(2): 340–352.
8. Ateniese G, Burns RC, Curtmola R, Herring J, Kissner L, Peterson ZNJ, Song DX. Provable data possession at untrusted stores. *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS'07*, Alexandria, VA, USA, 2007; 598–609.
9. Juels A, Jr BSK. Pors: proofs of retrievability for large files. *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS'07*, Alexandria, VA, USA, 2007; 584–597.
10. Dimakis AG, Godfrey B, Wainwright MJ, Ramchandran K. Network coding for distributed storage systems. *26th IEEE International Conference on Computer Communications, INFOCOM'07*, Anchorage, Alaska, USA, 2007; 2000–2008.
11. Halevi S, Harnik D, Pinkas B, Shulman-Peleg A. Proofs of ownership in remote storage systems. *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS'11*, Chicago, Illinois, USA, 2011; 491–500.
12. Bellare M, Keelveedhi S, Ristenpart T. Message-locked encryption and secure deduplication. *Advances in Cryptology - EUROCRYPT'13*, Athens, Greece, 2013; 296–312.
13. Ateniese G, Burns RC, Curtmola R, Herring J, Khan O, Kissner L, Peterson ZNJ, Song D. Remote data checking using provable data possession. *ACM Transactions on Information and System Security* 2011; **14**(1): 1-34.
14. Erway CC, Küpçü A, Papamanthou C, Tamassia R. Dynamic provable data possession. *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS'09*, Chicago, Illinois, USA, 2009; 213–222.
15. Wang C, Wang Q, Ren K, Lou W. Privacy-preserving public auditing for data storage security in cloud computing. *29th IEEE International Conference on Computer Communications, INFOCOM'10*, San Diego, CA, USA, 2010; 525–533.
16. Yuan J, Yu S. Efficient public integrity checking for cloud data sharing with multi-user modification. *2014 IEEE International Conference on Computer Communications, INFOCOM'14*, Toronto, Canada, 2014; 2121–2129.
17. Wang H, He D, Tang S. Identity-based proxy-oriented data uploading and remote data integrity checking in public cloud. *IEEE Transactions on Information Forensics & Security* 2016; **11**(6): 1165–1176.
18. Bowers KD, Juels A, Oprea A. HAIL: a high-availability and integrity layer for cloud storage. *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS'09*, Chicago, Illinois, USA, 2009; 187–198.
19. Xu J, Chang EC. Towards efficient proofs of retrievability. *7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS'12*, Seoul, Korea, 2012; 79–80.
20. Shacham H, Waters B. Compact proofs of retrievability. *Journal of Cryptology* 2013; **26**(3): 442–483.
21. Armknecht F, Bohli JM, Karame GO, Liu Z, Reuter CA. Outsourced proofs of retrievability. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS'14*, Scottsdale, AZ, USA, 2014; 831–843.
22. Cash D, Küpçü A, Wichs D. Dynamic proofs of retrievability via oblivious RAM. *Advances in Cryptology - EUROCRYPT'13*, Athens, Greece, 2013; 279–295.
23. Shi E, Stefanov E, Papamanthou C. Practical dynamic proofs of retrievability. *2013 ACM SIGSAC*

Conference on Computer and Communications Security, CCS'13, Berlin, Germany, 2013; 325–336.
24. Ren Z, Wang L, Wang Q, Xu M. Dynamic proofs of retrievability for coded cloud storage systems. *IEEE Transactions on Services Computing* 2015; **PP**(99): 1–1.
25. Di Pietro R, Sorniotti A. Boosting efficiency and security in proof of ownership for deduplication. *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ASIACCS'12*, Seoul, Korea, 2012; 81–82.
26. Zheng Q, Xu S. Secure and efficient proof of storage with deduplication. *2nd ACM Conference on Data and Application Security and Privacy, CODASPY'12*, San Antonio, TX, USA, 2012; 1–12.
27. Stefanov E, Dijk Mv, Juels A, Oprea A. Iris: a scalable cloud file system with efficient integrity checks. *28th annual computer security applications conference, ACSAC'12*, Orlando, FL, USA, 2012; 229–238.
28. Chen J, He K, Du R, Zheng M, Xiang Y, Yuan Q. Dominating set and network coding-based routing in wireless mesh networks. *IEEE Transactions on Parallel and Distributed Systems* 2015; **26**(2): 423–433.
29. Chen J, He K, Yuan Q, Du R, Wu J. Distributed greedy coding-aware deterministic routing for multi-flow in wireless networks. *Computer Networks*, DOI: 10.1016/j.comnet.2016.05.027.
30. Ho T, Médard M, Koetter R, Karger DR, Effros M, Shi J, Leong B. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory* 2006; **52**(10): 4413–4430.