Proofs of Physical Reliability for Cloud Storage Systems

Li Li and Loukas Lazos
Dept. of Electrical and Computer Engineering, The University of Arizona
{lili, llazos}@email.arizona.edu

Abstract—Cloud service providers (CSPs) promise to reliably store repositories outsourced by clients. Unfortunately, once files have left the client's control, he has no means to verify their redundant storage. In this paper, we develop Proof of Physical Reliability (PoPR) auditing mechanisms that prove that a CSP stores an outsourced repository across multiple physical storage nodes. A PoPR complements the existing proof-of-retrievability (PoR) and proof-of-data possession (PDP) methods that are concerned with file retrievability, but without any verification of the fault-tolerance to physical storage nodes failures. A PoPR goes beyond retrievability by verifying that a file is redundantly stored across multiple physical storage nodes according to a pre-agreed layout and can, therefore, survive node failures. The verification mechanism relies on a combination of storage integrity and timing tests on the simultaneous retrieval of a collection of file symbols from multiple storage nodes. Compared to the state-of-the-art, our approach accommodates CSPs with heterogeneous storage devices (hard disks, SSDs, etc.) and does not assume constant data processing nor network delays. Instead, it can operate under any delay variance, because it relies only on (loose) delay bounds. We analytically prove the security of our construction and experimentally validate its success in heterogeneous storage settings.

Index Terms—Proof of reliability, fault tolerance, data integrity, data security and privacy, storage reliability, retrievability.

1 Introduction

With the continuously decreasing costs of cloud services, outsourcing large repositories online is becoming the norm. Recent studies reported that over one exabyte of data from more than 50% of large enterprises is already stored in the cloud [1]. However, the lack of *provable reliability and security guarantees* exposes cloud users–enterprises, government, public institutions, individuals–to legal, financial, and business liabilities [2]. As a result, sensitive data from the healthcare, finance, defense, and other risk-averse sectors is still maintained in-house with high costs.

To mitigate data loss and privacy risks, enterprises negotiate contractual terms with Cloud Service Providers (CSPs), reflected in Service Level Agreements (SLAs). An SLA outlines data availability guarantees against Byzantine failures, misconfigurations, attacks, and any other disruption. However, SLAs do not specify mechanisms for verifying the adherence to the SLA terms [2]. Accidental misconfigurations may lead to irrecoverable data losses [3]. Furthermore, economically motivated CSPs may choose to circumvent the promised level of service to reduce their storage, network, and other operational costs. Possible actions include the deletion of rarely accessed files [4]–[11], distribution of files across fewer storage nodes [12]–[15], and data replication to fewer geographic locations [16]–[19]. To protect clients against SLA violations, researchers have introduced storage integrity verification protocols [4]–[10], [20]–[23]. The so called Proof-of-Retrievability (PoR) and Proof-of-Data Possession schemes invoke interactive protocols between the CSP and the client (or a trusted verifier) to ensure data retrievability [4], [10], [22], [23] or that it is stored redundantly [11], [19], [24]–[26]. However, prior works on PoR or PDP only verify the existence of data at the logical level. They offer no proof on how the data is physically distributed across multiple storage nodes.

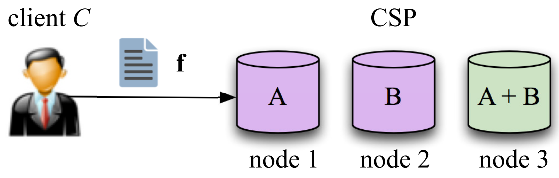

Fig. 1: File $\mathbf{f}$ is encoded to symbols A, B, and $A+B$ and stored in three storage nodes.

We address the problem of physical storage integrity verification, where the distribution of an outsourced repository across multiple physical storage nodes is verified. To motivate the need for such service, consider the basic scenario shown in Fig. 1. A client C outsources a file $\mathbf{f}$ to the CSP, who agrees to store $\mathbf{f}$ such that it is recoverable from any single storage node failure. For this purpose, the CSP applies a linear error correction code (ECC) $\mathcal{C}$ on $\mathbf{f}$. In our example, $\mathbf{f}$ is divided to two symbols A and B and the application of the ECC yields one parity symbol $A+B$. A PoR or a PDP can verify that the CSP stores A, B, and $A+B$, such that the file is still retrievable if any one symbol is erased. However, the PoR/PDP test cannot verify how the sybmols are distributed across storage nodes. A dishonest CSP can choose to store all three symbols on the same storage node to save on network bandwidth if some symbol needs to be repaired from the other two. Failure of that storage node, however, renders the file irrecoverable. *In this paper, we aim at developing auditing mechanisms to verify that symbols of the encoded file are stored across separate physical storage nodes so that resilience to node failures is guaranteed. We refer to such* *mechanisms as Proofs-of-Physical-Reliability* (PoPR).

Verification of fault tolerance to storage node failures was first introduced by Bowers *et al.* in [15]. The authors proposed the RAFT protocol in which proof of physical storage was achieved by measuring the response time of requests on retrieving a collection of symbols. The queried symbols were distributed across a number of drives (acting as storage nodes) according to a known layout. If the CSP violated that layout by storing file symbols at fewer nodes, the response time would exceed the response deadline. However, RAFT could only verify storage on the same class of nodes, enterprise hard drives in particular, in which the drive read delay (time needed for the drive to seek and read a symbol) is relatively constant and known to the verifier. Moreover, the network latency was assumed to exhibit little variance, so that the response deadline could be set proportionally to the number of symbols retrieved. Similar assumptions were made in a follow-up work [13]. In practice, heterogeneous storage media with varying symbol retrieval times may store $\mathbf{f}$. For instance, the read delay of solid state drives (SSDs) is several orders of magnitude smaller than that of rotational hard drives. Moreover, the network delay varies from query to query.

More recently, Vasilopoulos *et al.* introduced the POROS scheme for verifying cloud storage reliability [24]. The main idea of POROS is to use a PDP scheme to verify the data storage by the CSP and employ a timing test to detect the on-the-fly reconstruction of redundancy. However, POROS only verifies reliability at the logical level and does not provide any guarantees on the data distribution across storage nodes. Moreover, its security relies on the technical characteristics of rotational hard drives. A follow-up protocol named PORTOS aims at remedying these shortcomings by performing parallel PDP checks at several storage nodes, which must adhere to a set of deadlines [27]. PORTOS incorporates time-lock puzzles that impose a minimum delay (via computation) with every symbol retrieval. Thus, a dishonest CSP violating the logical and physical storage reliability is detected. Unfortunately, the security of PORTOS relies on the assumption that the computational capability of the CSP is known. To this date, there is no method to verifiably measure the computational capability of a CSP.

Contribution. We make the following contributions:

- We develop an interactive PoPR protocol that allows a verifier V attest that the CSP stores a file $\mathbf{f}$ across n distinct physical storage nodes, according to an agreed storage layout. Our method relies on a series of integrity and timing verification tests in which the CSP is requested to retrieve a collection of symbols from $\mathbf{f}$ in parallel.
- The PoPR protocol verifies that the CSP can tolerate up to t storage node failures, where t is the fault tolerance of the ECC applied to the outsourced data.
- We present two variations of the PoPR protocol that address two key scenarios. In the first scenario, the client (verifier) is assumed to store a local copy of the audited file and can therefore verify the integrity of the CSP's replies at no additional storage cost to the CSP. In the second scenario, no local copy is stored at the client. We develop a novel verification mechanism that relies on probabilistic spot checking to verify the storage integrity.
- Compared to prior works directly related to ours [13], [15], [24], [27], the security of our PoPR protocol does not rely on a specific storage technology. Rather, our scheme is compatible with a variety of storage media such as SSDs, HDDs, etc. and can be used to verify the distribution of data across heterogeneous storage nodes. Moreover, we do not place any computational restrictions on the CSP. Security is preserved even if the CSPs employs multi-core systems to speed-up any required computation. Finally, our protocol tolerates variance in network latency and does not need to be invoked from locations close to the audited data centers.
- We present a comprehensive security analysis of the proposed protocol and show it achieves an arbitrarily small false negative rate and a zero false positive rate. To account for the heterogeneous storage and network conditions, we rely on loose bounds on processing (seek and read) delay and network latency. These bounds reflect the limitations of existing technologies and can be adjusted to any future technology updates. We validate the correctness and soundness of our approach with testbed experiments over heterogeneous storage nodes that exhibit drive read delays differences by a factor of 100.

Organization. Section 2 gives an overview of the related work. We present models and assumptions in Section 3. In Section 4, we introduce the PoPR construction. We analyze the security of the PoPR in Section 5. Section 6 presents experimental results that validate the security of the PoPR construction. In Section 7, we present a PoPR variation that does not require the storage of a local file copy at the verifier. We conclude in Section 8.

2 Related Work

Prior art on storage integrity verification can be categorized into three classes: (a) verification of data retrievability, (c) verification of data replication at different geographic locations, and (b) verification of reliability. We briefly describe the state-of-the-art for the first and second class and focus our attention on physical reliability verification, which is most relevant to our work.

Verification of data retrievability. In this problem, the CSP is challenged to prove that an outsourced file $\mathbf{f}$ is retrievable. Earlier works proposed interactive mechanisms, known as proofs of data possession (PDP) [9], [28] that can detect corruption of large portions of $\mathbf{f}$. PoRs enhance PDPs in that they provide highly compact evidence of the storage of $\mathbf{f}$ in a probabilistic fashion [4], [22], [23], [29], [30]. Moreover, the privacy of $\mathbf{f}$ relative to the CSP and the verifier was considered in [5]–[7]. The PoR efficiency, dynamic data update, outsourcing, and implementation in cloud environments were investigated in more recent works [25], [26], [31]–[35]. Interested reader is referred to a survey by Tan *et al.* for a comprehensive overview of the PoR/PDP literature [36]. Classical PoR formulations verify the existence of $\mathbf{f}$ without proving the distribution across multiple storage nodes. Without physical storage verification, a single node failure could render $\mathbf{f}$ irrecoverable.

Verification of storage geodiversity. An important class of verification methods targets the attestation of file replication across geographically-separated data centers [16]–[19]. The main idea is to execute a PoR from verifiers located close to each data center to minimize network latency. If the verifier does not receive a response within a deadline, the file is assumed to be absent from the particular data center. The geodiversity verification methods cannot be used to audit storage nodes within the same data center.

Verification of distributed storage across storage nodes. The integrity verification of storage across multiple physical nodes was first considered in [15] with the proposal of the RAFT protocol. RAFT is a challenge-response protocol that verifies the file retrievability even if up to t physical storage nodes have failed. In RAFT, symbols are requested to be retrieved in parallel from all nodes that store the file. If fewer storage nodes are used by a dishonest CSP, the deadlines set by the verifier are violated because the indicated symbols cannot be retrieved fast enough. The authors propose the *lock-step* protocol for progressively revealing the order of the audited symbols and forcing the sequential symbol retrieval. The use of small symbols (64KB) minimizes the read delay variance, which is key for determining the response deadline and guaranteeing correctness. This limits the applicability of RAFT to scenarios with invariant read and network delays. If the CSP employs some faster drives (e.g., SSDs), or a network path with smaller latency, the timing test can be easily defeated. With continuous advancements in storage technology, it is difficult to produce faithful timing models for drive read delays [37]. *The key difference of our work with RAFT is that our method can accommodate heterogeneous storage technologies and network settings with variable delay. Moreover, our improved PoPR protocol does not require the storage of a file copy at the verifier, allowing the audit of large repositories.*

A layout-free verification protocol that tests for even file-symbol distribution across a fixed number of drives was proposed in [13]. Similar assumptions were made about the constancy of the network latency and drive read delay. Some efforts have targeted the verification of the file storage across a set of n servers in the presence of an external adversary who progressively corrupts different servers [20], [38], [39]. An efficient PoR is performed to detect erasures at different servers. These file verification mechanisms, however, cannot be deployed to attest storage across storage nodes that are under the control of a single entity (CSP).

More recent storage integrity verification protocols have extended beyond retrievability to verifying storage reliability. The POROS scheme proposed by Vasilipoulos *et al.* [24] combines a PDP test with a timing test to verify that a CSP stores the original file plus redundant symbols. Simultaneously, it enables the file maintenance at the CSP without any client interaction. The main difference between POROS and our scheme is that POROS has no mechanism to verify that the data is distributed across multiple physical storage nodes. That is, it only verifies reliability at the logical level. In fact, POROS requires all redundant symbols to be stored at a single storage node. Two storage node failures can render the data irretrievable. Moreover, the security of the timing test hinges on the rotational hard drives model for predicting the data retrieval delay, and assumes very little variance on the network delay.

In a follow up work, Vasilopoulos *et al.* introduced the PORTOS scheme to remedy all the shortcomings of POROS [27]. In PORTOS, time-lock puzzles are used to prevent the CSP from computing parity symbols efficiently on-the-fly. The main idea here is to request the retrieval of information and parity symbols stored in t nodes in parallel and verify both the integrity of the symbols retrieved and the time of the response. A dishonest CSP who does not store redundant symbols or does not store them according to the layout, must solve k time-lock puzzles (where k is the number of information symbols in the ECC encoding) before he can compute parity symbols. This significantly increases the response delay to an audit and eventually leads to the violation of the timing deadline. As pointed out by the authors, the main issue with their approach is that time-lock puzzles must be tuned to the computational capacity of the CSP. To this date, there is no method that can securely verify the computational capacity of a cloud provider. Moreover, the repair process is inefficient as the CSP must solve k puzzles every time a single symbol has to be recovered, thus inducing unnecessary computational burden to an honest CSP. In our work, we do not make any assumptions about the computational capacity of the CSP.

Summary of differences with most relevant work: Our PoPR construction is directly comparable to the RAFT scheme [15], the scheme in [13], POROS [24] and PORTOS [27]. With respect to the first three schemes, we do not require that the CSP uses HDDs with fixed seek and read delays. Our scheme can verify the storage integrity across n storage nodes, even when these nodes are heterogeneous (e.g., mix of HDDs and SSDs). Moreover, our scheme can tolerate variable network delay. Compared with PORTOS, we do not place any computational bound on the CSP. Such bounds are difficult to derive and verify in practice. There are other subtle differences with prior works that make our approach more practical. For instance, RAFT requires that the client stores a copy of the outsourced file for verification, whereas in our case, the client does not need to keep a local copy. In fact, the audit process can be outsourced to a third-party verifier. In addition, the POROS scheme cannot guarantee the storage reliability beyond two storage nodes (all parity symbols are stored on a single node).

3 Models and Assumptions

We consider a setting in which a client C outsources a repository $\mathbf{f}$ to the CSP. The CSP and C agree on an SLA which specifies that storage of $\mathbf{f}$ must survive the failure of any d_0 storage nodes. We refer to d_0 as the *fault-tolerance degree* of the PoPR. The client periodically audits the CSP with a challenge Q to prove that $\mathbf{f}$ is stored in at least d_0 storage nodes. The CSP must reply with a valid response R within a specified deadline.

3.1 Notation

Table 1 presents the most commonly used notation through the rest of the paper. We use boldface letters to denote any vector of symbols or vectors. For instance, a vector $\mathbf{x}$ may be partitioned to $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ sub-vectors, with each sub-vector $\mathbf{x}_i$ consisting of $\mathbf{x}_i = \{x_1, x_2, \ldots, x_n\}$ symbols.

TABLE 1: Commonly Used Notation.

n	The number of storage nodes
SN_i	The i^{th} storage node
$\mathbf{f} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n_s}\}$	A file $\mathbf{f}$ partitioned into n_s subfiles
$\mathbf{f}_i = \{f_{i,1}, f_{i,2}, \ldots, f_{i,k}\}$	The i^{th} subfile $\mathbf{f}_i$ containing k symbols
$f_{i,j}$	The j^{th} symbol in the i^{th} subfile
$\mathbf{c}$	The verifiable version of $\mathbf{f}$
$\mathbf{s}_i$	The set of symbols stored at SN_i
Q	A challenge constructed by the verifier
$\mathbf{x}_i$	The set of symbols challenged from SN_i
$a_{j,i}$	The index of the j^{th} symbol in $\mathbf{x}_i$
y_i	The cardinality of $\mathbf{x}_i$
r_i	The response generated by the CSP on $\mathbf{x}_i$
R = $\{r_1, \ldots, r_n\}$	The set of responses
t_i	The response time of r_i
$\tau(y_i)$	The deadline set for r_i, based on y_i
$T^{(P)}$	Per symbol processing delay
$T^{(P)}_{\max}$, $T^{(P)}_{\min}$	The upper and lower bounds of $T^{(P)}$
$T^{(N)}$	Network latency
$T^{(N)}_{\max}$, $T^{(N)}_{\min}$	The upper and lower bounds of $T^{(N)}$
η, η'	Random nonce

3.2 CSP Model

Storage model. Without loss of generality, we analyze the audit of single file $\mathbf{f}$ out of an entire repository. Multiple files can be audited independently, using separate audits. The file $\mathbf{f}$ is stored at a collection of heterogeneous storage nodes in a remote data center. To efficiently and reliably store large files, $\mathbf{f}$ can be partitioned to n_s subfiles of equal size. Each subfile $\mathbf{f}_i$ is encoded into a verifiable version $\mathbf{c}_i$ that incorporates redundancy through the application of a linear error correction code (ECC) $\mathcal{C}$ of length n, rank k, and minimum distance d over a finite field $\mathbb{F}_q$, an $(n, k, d)_q$ code for short. The order of $\mathbb{F}_q$ is $q = 2^\ell$, so code $\mathcal{C}$ operates on ℓ-bit blocks, referred to as *symbols*. This is a common practice in most cloud systems [40]–[42]. The encoded file $\mathbf{c} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{n_s}\}$ is stored according to a layout $\mathcal{L}$ that satisfies the agreed fault-tolerance degree d_0. We note that $d_0 \leq d$, i.e., the required fault-tolerance degree cannot exceed the code distance. For simplicity, we map symbol redundancy to storage node redundancy by storing $\mathbf{c}$ at n nodes in the layout shown in Fig. 2. The symbols $\mathbf{c}_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,n}\}$ of each subfile $\mathbf{c}_i$ are dispersed across the n storage nodes. Therefore, each node SN_i stores one symbol from each subfile. The set of symbols stored at SN_i denote a file sector $\mathbf{s}_i = \{c_{1,i}, c_{2,i}, \ldots, c_{n_s,i}\}$. The storage layout $\mathcal{L}$ is known to the verifier.

The CSP uses a parallel I/O system to retrieve symbols from different storage devices. The degree of parallelism equals the number of nodes used to store $\mathbf{f}$. Symbols on the same storage device are retrieved serially.

Timing model. The proposed PoPR method is an interactive timed challenge-response protocol that measures the response time between the submittal of a challenge Q by the verifier V and the reception of the response R. The response time is composed of the processing delay $T^{(P)}$ for constructing R and the network latency $T^{(N)}$ for Q and R.

Processing delay per symbol $T^{(P)}$: The processing delay consists of the read delay for retrieving a symbol from a physical storage device and the computation time for compiling a response.

$$T^{(P)} = T^{(R)} + T^{(C)} \tag{1}$$

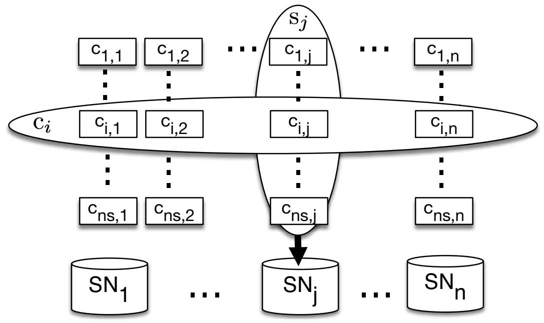

Fig. 2: The storage layout for encoded file $\mathbf{c}$.

Here, we do not assume any particular model for the read time $T^{(R)}$ and computation time $T^{(C)}$. This generalizes our method to any storage technology. The only assumption made is that the processing delay to retrieve and compute on one symbol (unit of data) is unknown but bounded. Upper and lower bounds are defined by

$$T^{(P)}_{\min} < T^{(P)} < T^{(P)}_{\max}, \tag{2}$$

where $T^{(P)}_{\min}$ and $T^{(P)}_{\max}$ denote the slowest and fastest delays for processing one symbol, respectively. The time bounds combine both data retrieval and computation.

We emphasize here that the CSP could use a multi-core system to speed up computation and minimize $T^{(C)}$. Even if the CSP has unbounded computational power ($T^{(C)} = 0$), the processing delay is still bounded by the delay $T^{(R)}$ for serially retrieving symbols from a physical storage device. The latter delay cannot be sped-up due to the physical hardware limitations and also due to the serial retrieval process imposed by our protocol. Note that the time bounds need not be tight. For instance, the upper bound can be set based on the fastest data access rate known by today's technological standards plus some error margin, whereas the CSP can commit to a slowest processing rate. Looser bounds ensure correctness at the expense of longer response times and the retrieval of more symbols.

Network latency $T^{(N)}$: To make our protocol practical for any network setting, we do not adopt a precise model for the network latency. Similar to the processing delay, we consider the network latency (transmission plus propagation plus queuing delay) for sending and receiving (round trip time) one unit of data to be unknown but bounded by

$$T^{(N)}_{\min} < T^{(N)} < T^{(N)}_{\max}, \tag{3}$$

where $T^{(N)}_{\min}$ and $T^{(N)}_{\max}$ denote the lower and upper bounds on the network latency. These bounds are conservative estimates based on statistical estimates and need not be tight. For instance, given a sample distribution of network delays, the lower and upper bounds can be set to several standard deviations away from the expected value or the min and max of the samples plus/minus some error margin.

3.3 Adversary Model

The adversary is an untrusted CSP who alters the pre-agreed storage layout $\mathcal{L}$ to store data on fewer storage nodes. This strategy reduces the network overhead for maintaining storage, which is a significant data center cost [41]. The CSP is considered to be rational in that he will alter the storage layout only if he could successfully pass the verifier's challenges. Moreover, he is cheap and lazy in the sense that he will not attempt to alter the stored symbols or split a symbol among multiple drives. The symbol obliviousness abstraction allows us to consider symbols as the smallest atomic unit. No restrictions are placed on the way that the symbols of $\mathbf{f}$ are redistributed by the CSP. Some could be replicated and stored in multiple storage nodes, whereas others could be deleted. Finally, the CSP does not gain any advantage by re-arranging the layout $\mathcal{L}$ without reducing the number of storage nodes.

4 The PoPR Construction

In this section, we describe our PoPR construction which verifies the storage of $\mathbf{f}$ according to the layout $\mathcal{L}$. For simplicity, we focus on a single file $\mathbf{f}$, but our methods extend to a file repository in a straightforward manner.

4.1 PoPR Overview

Consider a client that wants to outsource a file $\mathbf{f}$ to the CSP. The proposed PoPR construction consists of two phases: the setup phase and the verification phase. In the setup phase, the client transforms $\mathbf{f}$ to a verifiable version $\mathbf{c}$. The transformation includes the partition of $\mathbf{f}$ to subfiles of size k symbols and the application of an $(n, k, d)_q$ code $\mathcal{C}$ that can tolerate up to d symbol erasures. The symbols of each subfile are then distributed to n physical storage nodes (one symbol per node for each codeword) to tolerate up to d storage node failures. This creates a storage layout arrangement $\mathcal{L}$ that is agreed between the client and the CSP.

In the verification phase, the verifier performs an integrity and timing test to validate the number of nodes that store $\mathbf{c}$. Initially, the verifier constructs a challenge Q by choosing the number of symbols to query from each storage node. He also chooses the corresponding deadlines that the CSP's responses from each node are expected. The deadlines assume that the CSP will retrieve the indicated symbols in parallel, because they are stored at n nodes. If the CSP violates the agreed layout $\mathcal{L}$ and stores symbols to fewer nodes, he would fail to meet all the deadlines because more symbols have to be retrieved from fewer drives. The integrity test ensures that the CSP constructs his response by indeed retrieving the designated symbols.

To demonstrate the main idea, consider the 2-replication of $\mathbf{f}$ at two storage nodes SN_1 and SN_2. That is, copy $\mathbf{s}_1$ of $\mathbf{f}$ is stored at SN_1 and a second copy $\mathbf{s}_2$ is stored at SN_2. To verify the physical storage of $\mathbf{s}_1$ and $\mathbf{s}_2$ at two separate nodes, the verifier selects at random two sets of symbols to be challenged from each storage node. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ denote the collection of challenged symbols at SN_1 and SN_2, respectively. The verifier issues a challenge Q that defines the symbol indices for $\mathbf{x}_1$ and $\mathbf{x}_2$ and a random nonce η. Upon receiving challenge Q, the CSP retrieves $\mathbf{x}_1$ and $\mathbf{x}_2$ to construct a response $R = \{r_1, r_2\}$. The CSP does not wait until both $\mathbf{x}_1$ and $\mathbf{x}_2$ are retrieved, but sends to the verifier each of r_1 and r_2 as soon as they become available. The verifier validates the integrity of the responses and compares the response times t_1 and t_2 with preset deadlines.

The sizes of $\mathbf{x}_1$ and $\mathbf{x}_2$ are selected in such a way that if $\mathbf{x}_1$ and $\mathbf{x}_2$ are retrieved in parallel, the CSP response times satisfy the selected deadlines, whereas serial retrieval from the same storage node violates them. The key step is to tune the sizes of $\mathbf{x}_1$ and $\mathbf{x}_2$ in such a way that even if a single faster storage media is used, at least one of the deadlines is violated. This idea directly extends to the n storage nodes. We now describe the setup and verification phases in detail. We provide a basic protocol that assumes that $\mathbf{f}$ is stored at the verifier. We then construct a protocol that does not require the maintenance of a local file copy at the verifier.

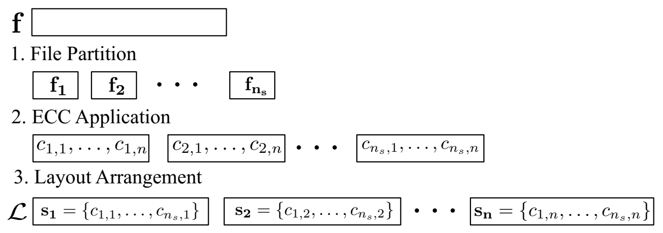
$\mathbf{f}$
1. File Partition
$\mathbf{f_1}$ $\mathbf{f_2}$ $\cdots$ $\mathbf{f_{n_s}}$
2. ECC Application
$c_{1,1}, \ldots, c_{1,n}$ $c_{2,1}, \ldots, c_{2,n}$ $\cdots$ $c_{n_s,1}, \ldots, c_{n_s,n}$
3. Layout Arrangement
$\mathcal{L}$ $\mathbf{s_1} = \{c_{1,1}, \ldots, c_{n_s,1}\}$ $\mathbf{s_2} = \{c_{1,2}, \ldots, c_{n_s,2}\}$ $\cdots$ $\mathbf{s_n} = \{c_{1,n}, \ldots, c_{n_s,n}\}$

Fig. 3: Setup phase.

4.2 Setup Phase

In the setup phase, the client encodes $\mathbf{f}$ to a verifiable version $\mathbf{c}$ by applying the following steps:

1) **File partition.** The file $\mathbf{f}$ is divided to n_s subfiles $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n_s}$. Each subfile has k symbols. If $\mathbf{f}$ is not a multiple of k, then it is padded.
2) **ECC application.** Each $\mathbf{f}_i$ is encoded to $\mathbf{c}_i$ by applying an $(n, k, d)_q$ code $\mathcal{C}$, with $\mathbf{c}_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,n}\}$.
3) **Layout arrangement.** A layout $\mathcal{L} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n\}$ is created by arranging $\mathbf{c}$ to n sectors, with each sector $\mathbf{s}_i = \{c_{1,i}, c_{2,i}, \ldots, c_{n_s,i}\}$ hosting one symbol per encoded subfile $\mathbf{c}_i$. Each sector hosts n_s symbols.
4) **File upload.** The encoded file $\mathbf{c}$ and the layout $\mathcal{L}$ are provided to the CSP. The CSP stores $\mathbf{c}$ according to $\mathcal{L}$ in n storage nodes.

The steps of the setup phase are shown in Fig. 3. At the end of the setup phase, both the client and the CSP agree on $\mathcal{L}$. Note here that for simplicity, we have assumed that the number of nodes used to store $\mathbf{c}$ is equal to the code length. However, this is not a requirement. A smaller number of nodes can be used for storage at the expense of storage reliability. In practical scenarios, the code length n should be selected to be equal to the number of storage nodes used at the CSP to guarantee resilience to up to d failures.

4.3 Verification Phase

The verification phase is an interactive protocol between the verifier and the CSP that verifies the storage layout $\mathcal{L}$ at the CSP. This is achieved probabilistically, by auditing a subset of symbols from each node. The verification phase consists of the construction of challenge Q by the verifier and the response R by the CSP.

Construction of Challenge Q.

1) **Deadline selection.** The verifier specifies the number $y_1, y_2, \ldots, y_n$ of symbols to be queried from each storage node (sector). The verifier also selects n deadlines $\tau(y_1), \tau(y_2), \ldots, \tau(y_n)$.

2) **Initial symbol.** The verifier selects a random nonce η and n random indices $a_{1,1}, a_{1,2}, \ldots, a_{1,n}$, with $a_{j,i} \in [1, n_s]$. The random index $a_{1,i}$ specifies the first symbol $x_{i,1} = c_{(a_{1,i}),i}$ to be queried from $\mathbf{s}_i$.
3) **Query construction.** The verifier applies a public pseudo-random function $\phi : \{0,1\}^{\ell} \times \{0,1\}^* \to [1, n_s]$, such as a strong universal hash function, y_i times to find the last symbol to be retrieved from sector $\mathbf{s}_i$. Here $a_{j+1,i} = \phi(c_{(a_{j,i}),i}, \eta)$, i.e., the index of the next symbol to be retrieved is computed based on the previously retrieved symbol and η.
4) **Challenge construction.** The verifier selects a random nonce η' and computes hash values $h(c_{(a_{y_i,i}),i}, \eta')$. The challenge Q consists of $\{a_{1,1}, a_{1,2}, \ldots, a_{1,n}\}$, $h(c_{(a_{y_1,1}),1}, \eta')$, $h(c_{(a_{y_2,2}),2}, \eta'), \ldots, h(c_{(a_{y_n,n}),n}, \eta')$, η, and η', where $h(\cdot)$ is a secure hash function, and η' is a nonce. The verifier sends Q to the CSP and initializes n clocks $t_1, t_2, \ldots, t_n$ to zero.

As an example, consider a file $\mathbf{f}$ stored in two storage nodes SN_1 and SN_2. Let the sectors $\mathbf{s}_1$ and $\mathbf{s}_2$ store 9 symbols each. In step 1, the verifier specifies the number of symbols that need to be retrieved for the timing test to be equal to 3 symbols from each sector. In step 2, the verifier selects two random indices from 1 to 9. Say $a_{1,1} = 2$ and $a_{2,1} = 3$. The verifier selects a nonce η and computes $\phi(c_{(a_{j,i}),i}, \eta)$. Let the indexes computed via pseudo-random function ϕ be $a_{2,1} = 5$ and $a_{3,1} = 9$ for sector $\mathbf{s}_1$ and $a_{2,2} = 7$ and $a_{3,2} = 8$ for $\mathbf{s}_2$. The symbols to be retrieved form $\mathbf{s}_1$ are $\mathbf{x}_1 = \{c_{2,1}, c_{5,1}, c_{9,1}\}$ and from $\mathbf{s}_2$ are $\mathbf{x}_2 = \{c_{3,2}, c_{7,2}, c_{8,2}\}$. The challenge Q consists of the initial indices $\{2, 3\}$, the hashes of the ending symbols with random nonce η', i.e., $\{h(c_{9,1}, \eta'), h(c_{8,2}, \eta')\}$, η, and η'. The symbol selection process for $\mathbf{x}_1$ is shown in Fig. 4.

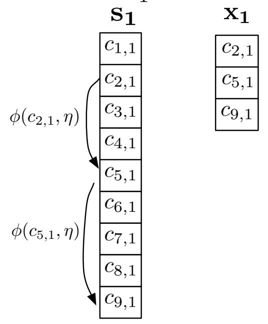

Fig. 4: Sequential symbol retrieval for $\mathbf{x}_1$.

The construction of Q achieves several critical properties for the security of the PoPR. First, each set $\mathbf{x}_i$ retrieved from a sector $\mathbf{s}_i$ is randomized with every challenge. This is achieved by randomly selecting the initial index $a_{i,1}$, incorporating the nonce η, and applying ϕ. This property ensures that symbols in each $\mathbf{x}_i$ are not sequential to account for possible reduction in data access times in sequential reading. Note that this is not necessary from a security standpoint because delay bounds are used for data access, but it nonetheless improves the margin of detection. Moreover, the sequence of queried symbols cannot be precomputed based on the initial index due to η.

A second important property is that the next symbol to be retrieved depends on the previous one through the pseudo-random function ϕ. This forces a serial retrieval process so that the data processing delay is cumulative. Finally, ϕ hides the number of symbols that are to be retrieved from every storage node. The end of retrieval process for SN_i is reached when the hash of the retrieved symbol matches $h(c_{(a_{y_i,i}),i}, \eta')$. This condition can only be tested if all y_i symbols have been retrieved. Hiding the length of each challenge is a prerequisite for the success of the timing test, as we show in Section 4.5.

Construction of the Response R and Verification

1) **Symbol retrieval.** The CSP retrieves the symbols in each $\mathbf{x}_i$ in parallel from each node, starting from $c_{(a_{1,i}),i}$ and ending at $c_{(a_{y_i,i}),i}$. Symbols at different storage nodes are retrieved in parallel. For each retrieved symbol $c_{(a_{j,i}),i}$, the CSP computes $h(c_{(a_{j,i}),i}, \eta')$ and compares it with $h(c_{(a_{y_i,i}),i}, \eta')$ provided by the verifier to determine the query end.
2) **Response.** Upon the collection of all symbols in $\mathbf{x}_i$, the CSP computes $r_i = h(h(c_{(a_{1,i}),i}, \eta'), h(c_{(a_{2,i}),i}, \eta'), \ldots, h(c_{(a_{y_i,i}),i}, \eta'))$. The CSP sends each r_i to the verifier, as it becomes available.
3) **Verification.** The verifier records the arrival times for each response r_i and performs an integrity and a timing test. The CSP passes verification if both tests are passed.

The response R consists of n individual responses $r_1, r_2, \ldots, r_n$. Each response is constructed independently of the others. The symbols within the same storage node are retrieved sequentially, whereas the symbols across storage nodes are retrieved in parallel. The response r_i for a sector $\mathbf{s}_i$ consists of a hash of all hashed symbols in $\mathbf{x}_i$. Precomputation of the response is prevented by including the random nonce η' with every individual hash computation and by hiding the indices of symbols in each $\mathbf{x}_i$. The indices are also randomized with every repetition of the protocol. The responses r_i arriving at the verifier are checked for their integrity and timeliness by performing an integrity and timing test. These tests are meant to verify that the indicated symbols were retrieved and that the responses for each sector satisfied the specified deadlines.

4.4 Integrity Test

The integrity test checks the correctness of the response R returned by the CSP. This is necessary to ensure that the CSP indeed retrieved the intended symbols and did not create responses based on random symbol values. The verifier performs the integrity test according to the following steps:

1) **Response reception.** The verifier receives $R = \{r'_1, r'_2, \ldots, r'_n\}$ from the CSP.
2) **Response reconstruction.** The verifier retrieves each symbol $c_{(a_{j,i}),i}$ in $\mathbf{x}_i$ from the local copy $\mathbf{f}$ and computes $r_i = h(h(c_{(a_{1,i}),i}, \eta'), h(c_{(a_{2,i}),i}, \eta'), \ldots, h(c_{(a_{y_i,i}),i}, \eta'))$, for $i = 1..n$.
3) **Integrity check.** The CSP passes the integrity test if $r'_i = r_i$, for $i = 1..n$.

Note that the verifier can store $\mathbf{f}$ and recreate $\mathbf{c}$ on demand, since $\mathbf{c}$ is just an encoding of $\mathbf{f}$ with a public code.

The verifier can store the encoded version $\mathbf{c}$ to avoid this computation step at the cost of a storage overhead of n/k. The accessibility of $\mathbf{c}$ allows the verifier to check the correctness of each r_i and hence guarantee that actual symbols were retrieved once a correct r_i is received. In Section 7, we construct an integrity test that eliminates the storage of $\mathbf{f}$ at the verifier, at the expense of increased storage at the CSP.

We emphasize that the integrity test does not verify the retrievability of $\mathbf{f}$ at the CSP, but is limited to the retrieval of the $\mathbf{x}_i$'s. A PoR test can be performed separately for verifying the file retrieval. However, if the number of retrieved symbols in all $\mathbf{x}_i$ exceeds the number of symbols required for a PoR test [23], no separate PoR test is necessary.

4.5 Timing Test

The timing test verifies that the CSP conforms to the agreed storage layout. To perform a timing test, the verifier must select the lengths $y_1, y_2, \ldots, y_n$ of the n challenges and also set the deadlines $\tau(y_1), \tau(y_2), \ldots, \tau(y_n)$ for receiving responses $r_1, r_2, \ldots, r_n$ so that the deadlines are met only if the challenged symbols are retrieved in parallel from n storage nodes. If a dishonest CSP cheats by storing the symbols of $\mathbf{c}$ on fewer nodes, at least one of the n deadlines is violated. We first demonstrate how to select the challenge lengths and deadlines for two storage nodes and then extend to n nodes.

Selection of challenge lengths and deadlines. Consider the storage of $\mathbf{c}$ at SN_1 and SN_2 according to layout $\mathcal{L}$. Let the CSP be challenged to retrieve $\mathbf{x}_1$ of length y_1 from SN_1 and $\mathbf{x}_2$ of length y_2 from SN_2, in parallel. For a CSP that retrieves $\mathbf{x}_1$ and $\mathbf{x}_2$ in parallel from two storage nodes, the verifier should receive the responses r_1 and r_2 no later than

$$t_1 \leq \; y_1 T_{\max}^{(P)} + T_{\max}^{(N)} \tag{4}$$
$$t_2 \leq \; y_2 T_{\max}^{(P)} + T_{\max}^{(N)}, \tag{5}$$

where $T_{\max}^{(P)}$ is the upper bound on the per-symbol processing time (data retrieval plus hashing) and $T_{\max}^{(N)}$ is the upper bound on the network latency for receiving the challenge and sending each response of fixed length (size of a hash) back to the verifier. The upper bounds in (4) and (5) are based on the upper bounds in (2) and (3). The verifier sets the respective deadlines for receiving r_1 and r_2 to

$$\tau(y_1) = \; y_1 T_{\max}^{(P)} + T_{\max}^{(N)}, \tag{6}$$
$$\tau(y_2) = \; y_2 T_{\max}^{(P)} + T_{\max}^{(N)}. \tag{7}$$

The deadlines reflect the worst-case estimates on the response time of r_1 and r_2, assuming the longest processing delay and network latency. *Note that we do not assume a linear model for symbol processing.* The per-symbol processing delay may vary with every symbol retrieval. However, any per-symbol delay is upper bounded by $T_{\max}^{(P)}$, and hence, the inequalities (4) and (5) hold.

If a dishonest CSP stores $\mathbf{x}_1$ and $\mathbf{x}_2$ at a single storage node, both $\mathbf{x}_1$ and $\mathbf{x}_2$ have to be retrieved sequentially because of the challenge construction. Recall, that the previous symbol has to be retrieved before the next symbol in the challenge is determined. Without loss of generality, let the challenged symbols be stored in SN_1. The adversary is presented with three choices. Retrieve $\mathbf{x}_1$ first and then $\mathbf{x}_2$ (strategy 1), retrieve $\mathbf{x}_2$ first and then $\mathbf{x}_1$ (strategy 2), or retrieve $\mathbf{x}_1$ and $\mathbf{x}_2$ in an interleaving fashion (strategy 3).

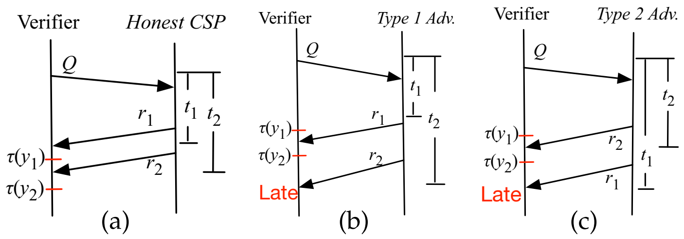

Fig. 5: Timeline for responses r_1 and r_2 when (a) $\mathbf{x}_1$ and $\mathbf{x}_2$ are stored at SN_1 and SN_2, respectively (honest strategy), (b) $\mathbf{x}_1$ and $\mathbf{x}_2$ are stored at SN_1 and $\mathbf{x}_1$ is retrieved first (strategy 1), and (c) $\mathbf{x}_1$ and $\mathbf{x}_2$ are stored at SN_1 and $\mathbf{x}_2$ is retrieved first (strategy 2).

Using the worst-case estimates for the honest strategy (storage of $\mathbf{x}_1$ and $\mathbf{x}_2$ at SN_1 and SN_2, respectively) and the best-case estimates for the dishonest strategies, we can derive the deadlines for which the timing test fails, when the storage layout is violated. For strategy 1,

$$(y_1 + y_2) T_{\min}^{(P)} + T_{\min}^{(N)} > \tau(y_2) \tag{8}$$

which, combined with (6), yields

$$y_1 > y_2 \left(T_{\max}^{(P)} / T_{\min}^{(P)} - 1 \right) + \left(T_{\max}^{(N)} - T_{\min}^{(N)} \right) / T_{\min}^{(P)}. \tag{9}$$

That is, the best-case response time of r_2 from a dishonest CSP should be longer than the worst-case response time of r_2 from an honest CSP. Following a similar approach for strategy 2, the verifier can detect that $\mathbf{x}_1$ and $\mathbf{x}_2$ are stored at the same storage node if

$$y_2 > y_1 \left(T_{\max}^{(P)} / T_{\min}^{(P)} - 1 \right) + \left(T_{\max}^{(N)} - T_{\min}^{(N)} \right) / T_{\min}^{(P)}. \tag{10}$$

For strategy 3, suppose the adversary retrieves $\mathbf{x}_1$ and $\mathbf{x}_2$ in an interleaving fashion and that the last retrieved symbol belongs to $\mathbf{x}_2$. The response time for $\mathbf{x}_1$ will be longer than if $\mathbf{x}_1$ was retrieved first entirety but it may meet the deadline. Meanwhile, the response r_2 cannot be constructed before both $\mathbf{x}_1$ and $\mathbf{x}_2$ are retrieved, because the last retrieved symbol belongs to $\mathbf{x}_2$. In this case, strategy 3 becomes equivalent to strategy 1, in that it cannot speed up the delivery of r_2. Similarly, if the last retrieved symbol belongs to $\mathbf{x}_1$, strategy 3 becomes equivalent to strategy 2. This shows that interleaving the symbol retrieval cannot simultaneously reduce the response time for both challenges. Therefore, we only analyze strategies 1 and 2.

The timing diagrams for the honest response strategy, strategy 1, and strategy 2 are shown in Fig. 5. In strategy 1, the CSP misses the deadline for r_2 whereas in strategy 2, the CSP misses the deadline for r_1. Inequalities (9) and (10) form the following system:

$$\begin{cases} y_1 > y_2 \left(T_{\max}^{(P)} / T_{\min}^{(P)} - 1 \right) + \left(T_{\max}^{(N)} - T_{\min}^{(N)} \right) / T_{\min}^{(P)}, \\ y_2 > y_1 \left(T_{\max}^{(P)} / T_{\min}^{(P)} - 1 \right) + \left(T_{\max}^{(N)} - T_{\min}^{(N)} \right) / T_{\min}^{(P)}. \end{cases} \tag{11}$$

Adding the two inequalities in (11) on both sides, we have

$$y_1 + y_2 > \left(T_{\max}^{(P)} / T_{\min}^{(P)} - 1 \right)(y_1 + y_2) + 2 \left(T_{\max}^{(N)} - T_{\min}^{(N)} \right) / T_{\min}^{(P)} \tag{12}$$

Note that $T_{\max}^{(P)} / T_{\min}^{(P)} - 1$ and $\left(T_{\max}^{(N)} - T_{\min}^{(N)} \right) / T_{\min}^{(P)}$ are both positive. Hence, the system has positive solutions of y_1 and y_2 only when

$$\frac{T_{\max}^{(P)}}{T_{\min}^{(P)}} < 2\left(1 - \frac{T_{\max}^{(N)} - T_{\min}^{(N)}}{T_{\min}^{(P)}(y_1 + y_2)}\right). \tag{13}$$

The bound of $T_{\max}^{(P)}/T_{\min}^{(P)}$ in (13) can be approximated to two and become independent of the sizes y_1 and y_2 when the minimum time to process $y_1 + y_2$ symbols is significantly larger than the time difference between the maximum and the minimum network latency. This is satisfied if y_1 and y_2 are large or the network latency shows little variance, such that the processing delay dominates the overall response time. In any case, and without using the aforementioned approximation, the verifier can pick a size y_1 and then solve (13) for y_2 such that the CSP will violate one of the two deadlines in (6) and (7), independent of his retrieval strategy.

However, the inequality in (13) cannot always be satisfied. One way to observe this is to have $T_{\max}^{(P)} > 2T_{\min}^{(P)}$. That is, the fastest processing time is at least twice the slowest processing time. In this case, there are no feasible sizes y_1 and y_2 that simultaneously satisfy the two inequalities in (11). To address this case, we propose a probabilistic solution that relies on repeated challenges.

Probabilistic Detection of Storage Layout Violations. When the processing and network delay bounds do not satisfy (13), the verifier can choose to satisfy one of the two inequalities by choosing a value for y_1 and calculating y_2 (or vice versa). It is straightforward to show that a dishonest CSP who stores $\mathbf{c}$ at a single storage node can evade detection by serving the shortest challenge first.

For instance, consider for simplicity that the network delay is negligible, so that its impact on (11) is eliminated. Let $T_{\max}^{(P)} = 11$ms and $T_{\min}^{(P)} = 1$ms, so that the two inequalities in (11) cannot be simultaneously satisfied. Let the verifier choose $y_2 = 10$ symbols and compute $y_1 = 200$ to satisfy the first inequality in (11). The respective deadlines are set according to (6) and (7) to $\tau(y_1) = y_1 T_{\max}^{(P)} = 2200$ms and $\tau(y_2) = y_2 T_{\max}^{(P)} = 110$ms, respectively. Let the dishonest CSP store both y_1 and y_2 at a single storage node SN_1 that operates with the lowest processing delay $T_{\min}^{(P)}$. If the CSP serves y_1 first followed by y_2, the response time for y_1 will be at $t_1 = y_1 T_{\min}^{(P)} = 200\text{ms} < \tau(y_1)$. However, the deadline for y_2 is violated since $t_2 = (y_1 + y_2)T_{\min}^{(P)} = 210\text{ms} > \tau(y_2)$. If the CSP serves the shortest challenge first (i.e. $\mathbf{x}_2$), both deadlines are met since $t_1 = (y_1 + y_2)T_{\min}^{(P)} = 210\text{ms} < \tau(y_1)$, and $t_2 = y_2 T_{\min}^{(P)} = 10\text{ms} < \tau(y_2)$.

The key to detecting storage layout violations is to *hide the lengths of y_1 and y_2 so that the CSP has to randomly choose which challenge must be served first.* This yields a probability of violating the timing test equal to $1/2$. To drive the probability of detection to any desired value, the verifier performs repeated independent challenge-response tests with hidden challenge lengths. Because tests are independent (the sequence of symbols challenged at each round is pseudo-random) and the lengths of the challenges randomly satisfy one of the two inequalities in (11), the probability that a dishonest CSP passes α tests is $1/2^{\alpha}$, which is a negligible function. The problem of detecting storage layout violations reduces to the problem of hiding the challenge lengths.

Hiding the challenge lengths. In the construction of challenge Q, we showed that the length of $\mathbf{x}_i$ remain hidden until the last symbol from each set is retrieved. This is because the next symbol to be retrieved depends on the previous one and the nonce η through the pseudo-random function ϕ. This recursive relationship guarantees that the CSP cannot determine the length of each challenge by the time it receives Q. Moreover, the end of the challenge is determined only after the last symbol is retrieved and hashed with η' so that it can be compared with the hash value provided by the verifier.

Extending the Timing Test to n Storage Nodes. Consider now the general case where $\mathbf{c}$ is stored at n nodes, but the CSP violates the layout agreement by reducing the storage nodes to $m < n$. Let $\mathcal{W}$ denote the deleted set of nodes and $\mathcal{E}$ the remaining set. Consider the challenge $\mathbf{x}_i$ to one remaining storage node SN_i and the challenge $\mathbf{x}_j$ to a deleted node SN_j To construct the responses r_i and r_j and pass the timing test, the CSP can employ any of the three strategies presented for the case of two storage nodes. That is, the CSP can choose to serve $\mathbf{x}_i$ first, followed by $\mathbf{x}_j$, serve $\mathbf{x}_j$ first followed by $\mathbf{x}_i$ or interleave the retrieval of symbols from each query. We emphasize that regardless of strategy, the CSP must retrieve all symbols in $\mathbf{x}_j$ serially from all nodes storing $\mathbf{x}_j$, due to the dependence of the next symbol from the previous one through the function ϕ. We also note that the interleaving retrieval strategy would naturally lead to higher delay because of the coordination/communication that is necessary to interrupt one retrieval process and initiate another among nodes that are connected via network controllers[1]. For now, we focus on the case where the deleted date is moved to the fastest storage node. We express the deadline inequalities that need to be satisfied, such that the verifier detects a layout violation for $\mathbf{x}_i$ and $\mathbf{x}_j$. If the CSP chooses to retrieve $\mathbf{x}_i$ first followed by $\mathbf{x}_j$ then y_i and y_j have to satisfy inequality (14).

$$\begin{aligned} & y_j T_{\max}^{(P)} + T_{\max}^{(N)} < (y_i + y_j)T_{\min}^{(P)} + T_{\min}^{(N)} \\ & \qquad \Rightarrow y_i > y_j\left(T_{\max}^{(P)}/T_{\min}^{(P)} - 1\right) + \left(T_{\max}^{(N)} - T_{\min}^{(N)}\right)/T_{\min}^{(P)}. \end{aligned} \tag{14}$$

If the CSP chooses to retrieve $\mathbf{x}_j$ first followed by $\mathbf{x}_i$ then y_i and y_j have to satisfy inequality (15).

$$\begin{aligned} & y_i T_{\max}^{(P)} + T_{\max}^{(N)} < (y_i + y_j)T_{\min}^{(P)} + T_{\min}^{(N)} \\ & \qquad \Rightarrow y_j > y_i\left(T_{\max}^{(P)}/T_{\min}^{(P)} - 1\right) + \left(T_{\max}^{(N)} - T_{\min}^{(N)}\right)/T_{\min}^{(P)}. \end{aligned} \tag{15}$$

In a system where every storage node could be a deleted one, the inequalities for all $i, j \in \mathcal{N}, i \neq j$ are written as:

$$\begin{cases} y_j & > y_i\left(T_{\max}^{(P)}/T_{\min}^{(P)} - 1\right) + \left(T_{\max}^{(N)} - T_{\min}^{(N)}\right)/T_{\min}^{(P)} \\ y_i & > y_j\left(T_{\max}^{(P)}/T_{\min}^{(P)} - 1\right) + \left(T_{\max}^{(N)} - T_{\min}^{(N)}\right)/T_{\min}^{(P)}. \end{cases} \tag{16}$$

Similar to the case of two storage nodes, (16) has a valid solution, only when condition

$$\frac{T_{\max}^{(P)}}{T_{\min}^{(P)}} < 2\left(1 - \frac{T_{\max}^{(N)} - T_{\min}^{(N)}}{T_{\min}^{(P)}(y_i + y_j)}\right) \tag{17}$$

is met for all pairs of storage nodes and respective challenges y_i and y_j. In this case, a single test is sufficient to

1. The coordination/communication delay cost for interleaving between storage nodes can be minimized if they are directly accessible by the CPU. In this case, an interleaving strategy could possibly pass verification even if fewer storage nodes are used. However, this scenario has limited value for the CSP because he is no longer motivated to reduce the number of storage nodes to save on the network repair bandwidth.

determine the compliance of the CSP. When (17) can't be met for all pairs of y_i and y_j, a probabilistic detection approach has to be taken, where α independent tests are performed with hidden challenge lengths. A simple way to select the challenge lengths in this case would be use (18) in $\alpha/2$ of the tests and (19) in the other half. So that, for each pair of y_i and y_j, there must have one inequality in (16) being met. Hence, the probability that a dishonest CSP passes α tests remains to be no higher than $1/2^{\alpha}$.

$$\begin{cases} y_1 = & y_0 \\ y_i > & y_{i-1}(T^{(P)}_{\max}/T^{(P)}_{\min} - 1) + (T^{(N)}_{\max} - T^{(N)}_{\min})/T^{(P)}_{\min} \end{cases} \tag{18}$$

$$\begin{cases} y_n = y_0 \\ y_{i-1} > y_i(T^{(P)}_{\max}/T^{(P)}_{\min} - 1) + (T^{(N)}_{\max} - T^{(N)}_{\min})/T^{(P)}_{\min} \end{cases} \tag{19}$$

Although for most applications the number of storage nodes is expected to be relatively small, when the number of storage nodes increases, selecting the challenge lengths according to (18) or (19) could be problematic due to the increase in y_i by a factor of $(T^{(P)}_{\max}/T^{(P)}_{\min} - 1)$. An alternative audit approach is to perform pairwise challenges on storage node pairs and set the value of $\mathbf{y}$ in each challenge according to (11). For example, when there are 3 storage nodes, SN_1, SN_2 and SN_3, with $T^{(P)}_{\max} = 10T^{(P)}_{\min}$ and $T^{(N)}_{\max} = T^{(N)}_{\min} = 0$, instead of setting $y_1 = 10$, $y_2 = 100$ and $y_3 = 1000$ according to (18), the verifier can form 3 pairs, (SN_1, SN_2), (SN_1, SN_3) and (SN_2, SN_3), and then according to (11), set $y_1 = 10$ and $y_2 = 100$ in the challenge on SN_1 and SN_2, set $y_1 = 10$ and $y_3 = 100$ in the challenge on SN_1 and SN_3, and set $y_2 = 10$ and $y_3 = 100$ in the challenge on SN_2 and SN_3. Using this method, one verification on all n storage nodes requires $\binom{n}{2}$ challenges. Once the value of y_i is set, the corresponding deadline τ_{y_i} is set according to (20).

$$\tau(y_i) = y_i T^{(P)}_{\max} + T^{(N)}_{\max} \tag{20}$$

To demonstrate the deadline selection for more than two storage nodes, suppose there are three storage nodes and $T^{(P)}_{\max} = 70$ms, $T^{(P)}_{\min} = 50$ms, $T^{(N)}_{\max} = 8$ms, and $T^{(N)}_{\min} = 6$ms. Based on these values, (17) can be met for any positive y_i and y_j. One such selection is $y_1 = y_2 = y_3 = 18$. On the other hand, if $T^{(P)}_{\max} = 70$ms, $T^{(P)}_{\min} = 5$ms, $T^{(N)}_{\max} = 8$ms and $T^{(N)}_{\min} = 6$ms, there are no positive values of y_1, y_2, and y_3 that satisfy (17). For the latter scenario, the challenges on pairs (SN_1, SN_2), (SN_2, SN_3), and (SN_1, SN_3) have to be repeated with hidden challenge lengths as discussed before.

Interleaved retrieval strategy. To increase the chances of passing verification, the CSP could consider distributing the symbols of deleted nodes to the remaining nodes and retrieve the symbols in an interleaved fashion. For instance, assume that the CSP distributes the symbols stored at SN_i to the m remaining storage nodes. When challenged to prove the storage of $\mathbf{x_i}$, the CSP serially retrieves those symbols by interrupting the retrieval process at each of the remaining m nodes. We have argued that the required coordination and queuing of retrieval requests would make the retrieval process for $\mathbf{x}_i$ and the rest of $\mathbf{x}_j$'s far slower, but it has the advantage that fewer symbols need to be retrieved from each node compared to storing all deleted symbols of SN_i to a single node SN_j.

Though impractical, suppose there is a system that has zero additional time cost on coordinating the retrieval process in the m remaining nodes and also interrupting current retrieval for the rest of the challenges. It is straightforward to prove that when the symbols of a deleted node are distributed to m or fewer nodes, *there is one node that stores at least y_i/m symbols from any challenge $\mathbf{x_i}$, with length y_i*. This fact can be used to modify the inequality in (14) to achieve detection, even under zero coordination cost. The bound in (14) can be adjusted to

$$\begin{aligned} &y_j T^{(P)}_{\max} + T^{(N)}_{\max} < (\frac{1}{m} y_i + y_j) T^{(P)}_{\min} + T^{(N)}_{\min} \\ &\quad \Rightarrow y_i > m(y_j(T^{(P)}_{\max}/T^{(P)}_{\min} - 1) + (T^{(N)}_{\max} - T^{(N)}_{\min})/T^{(P)}_{\min}). \end{aligned} \tag{21}$$

The inequality in (21) allows the verifier to adjust the challenge lengths and deadlines to achieve detection. In a system where every storage node could be a deleted one, the inequalities for all $i, j \in \mathcal{N}, i \neq j$ are written as:

$$\begin{cases} y_j & > m(y_i(T^{(P)}_{\max}/T^{(P)}_{\min} - 1) + (T^{(N)}_{\max} - T^{(N)}_{\min})/T^{(P)}_{\min}) \\ y_i & > m(y_j(T^{(P)}_{\max}/T^{(P)}_{\min} - 1) + (T^{(N)}_{\max} - T^{(N)}_{\min})/T^{(P)}_{\min}). \end{cases} \tag{22}$$

Taking a closer look at (22), we note that the lengths y_i and y_j are increased by a factor of m compared to (16), thus increasing the communication overhead.

5 Security Analysis

In this section, we analyze the security of the PoPR protocol. We prove that (a) an honest CSP will always pass verification (correctness), and (b) a dishonest CSP fails verification with high probability when the storage layout is violated.

Proposition 1. *(Correctness) An honest CSP that satisfies the storage layout always passes verification.*

Proof. An honest CSP stores $\mathbf{c}$ according to the agreed layout $\mathcal{L}$. When challenged with Q, the CSP retrieves the symbols indicated in Q from each storage node and constructs the responses $r_i,\ i = 1..n$. Because the honest CSP stores $\mathbf{c}$, the responses provided by the CSP satisfy the integrity test. For the timing test, the CSP retrieves the symbols stored at each of the n nodes in parallel, starting from $c_{(a_{1,1}),1}, c_{(a_{1,2}),2}, \ldots, c_{(a_{1,n}),n}$ and ending at $c_{(a_{y_1,1}),1}, c_{(a_{y_2,2}),2}, \ldots, c_{(a_{y_n,n}),n}$. The processing delay $T^{(P)}_i$ of node SN_i is upper-bounded by $T^{(P)}_i < T^{(P)}_{\max}$ and the network delay $T^{(N)}_i$ is upper-bounded by $T^{(N)}_i < T^{(N)}_{\max}$. Therefore, each response r_i arrives at the verifier no later than $y_i T^{(P)}_i + T^{(N)}_i < \tau(y_i)$. That is, an honest CSP meets the deadlines for each storage node SN_i. □

Proposition 1 shows that our PoPR protocol achieves a zero false positive rate as long as the processing and network delays stay within the predefined bounds. Note that these bounds can be set loosely to guarantee that an honest CSP will always succeed in the timing test, despite any delay variance.

Proposition 2. *(Security) A CSP that violates the storage layout by deleting at least one storage node is detected with probability one when $T^{(P)}_{\max}/T^{(P)}_{\min} < 2\left(1 - (T^{(N)}_{\max} - T^{(N)}_{\min})/T^{(P)}_{\min}(y_i + y_j)\right)$ and* *probability $1 - 1/2^{\alpha}$ otherwise, where α is the number of challenge rounds executed by the verifier.*

Proof. When $T^{(P)}_{\max}/T^{(P)}_{\min} < 2\left(1 - (T^{(N)}_{\max}-T^{(N)}_{\min})/T^{(P)}_{\min}(y_i+y_j)\right)$, the verifier can select y_i and y_j to satisfy both inequalities in (16), $\forall i, j, i \neq j$. Without loss of generality, assume that the CSP has deleted all symbols from SN_i and distributed them on remaining storage nodes. Let SN_j be any one of the remaining storage nodes. To compute the responses, picks one of the two strategies, either retrieving $\mathbf{x}_i$ first and then $\mathbf{x}_j$ or retrieve $\mathbf{x}_j$ first and then $\mathbf{x}_i$. Recall that all $\mathbf{y}_i$'s satisfy the inequalities in (16). If the CSP chooses to serve $\mathbf{x}_i$ first, the deadline $\tau(y_j)$ is violated. That is,

$$\begin{aligned} t_j =&(y_i + y_j)T^{(P)}_j + T^{(N)}_j \\ >&(y_i + y_j)T^{(P)}_{\min} + T^{(N)}_{\min} \\ >&\left(y_j\left(T^{(P)}_{\max}/T^{(P)}_{\min} - 1\right) + (T^{(N)}_{\max}-T^{(N)}_{\min})/T^{(P)}_{\min} + y_j\right) T^{(P)}_{\min} \\ &+ T^{(N)}_{\min} \\ =&y_j T^{(P)}_{\max} + T^{(N)}_{\max} = \tau(y_j) \end{aligned} \tag{23}$$

If the CSP retrieves the symbols in $\mathbf{x}_j$ first, r_j will arrive on time but r_i will violate deadline $\tau(y_i)$. That is,

$$\begin{aligned} t_i =&(y_i + y_j)T^{(P)}_i + T^{(N)}_i \\ >&(y_i + y_j)T^{(P)}_{\min} + T^{(N)}_{\min} \\ >&\left(y_i + y_i\left(T^{(P)}_{\max}/T^{(P)}_{\min} - 1\right) + (T^{(N)}_{\max}-T^{(N)}_{\min})/T^{(P)}_{\min}\right) T^{(P)}_{\min} \\ &+ T^{(N)}_{\min} \\ =&y_i T^{(P)}_{\max} + T^{(N)}_{\max} = \tau(y_i) \end{aligned} \tag{24}$$

Hence, the deletion of a storage node is always detected.

When $T^{(P)}_{\max}/T^{(P)}_{\min} > 2\left(1 - (T^{(N)}_{\max}-T^{(N)}_{\min})/T^{(P)}_{\min}(y_i+y_j)\right)$, the verifier can select y_i and y_j such that only one of the inequalities in (16) is satisfied. Without loss of generality, suppose that SN_i is deleted and symbols of SN_i are redistributed on the remaining storage nodes. Let SN_j be one of the remaining storage nodes. Let also the verifier select y_i and y_j such that $y_j > y_i(T^{(P)}_{\max}/T^{(P)}_{\min} - 1) + (T^{(N)}_{\max}-T^{(N)}_{\min})/T^{(P)}_{\min}$. Without knowing if $y_1 > y_2$, the CSP chooses to serve $\mathbf{x}_i$ or $\mathbf{x}_j$ at random. Following the probabilistic detection of the storage layout violation analysis of Section 4.3, if the CSP serves the shortest query first, the responses r_i and r_j will not violate their respective deadlines. Serving $\mathbf{x}_j$ first, however, will lead to the violation of the deadline for r_i, according to (24).

Since the verifier chooses to satisfy one of the inequalities in (16) at random and the query lengths remain hidden from the CSP until the last symbol has been retrieved, the probability that the CSP violates a deadline equals $1/2$. As all challenges are constructed independently, a verification phase consisting of α independent challenges detects a layout violation with probability $1 - 1/2^{\alpha}$. □

Proposition 2 shows that the verifier can drive the probability of detecting a violation in the storage layout to any desired value using repeated queries, irrespective of the delay bounds. For certain bound relationship, the detection probability becomes one with only a single query.

6 Evaluation

In this section, we experimentally demonstrate that our PoPR construction can detect a dishonest CSP that violates the pre-agreed layout by storing the outsourced file to fewer nodes. We show that such a CSP will violate at least one deadline set by the verifier.

Setup: We considered a setup with three storage devices acting as the cloud storage nodes: a Seagate HDD_1 with 1.5TB capacity, a Hitachi HDD_2 with 320GB capacity and a Kingston SSD_1 with 120GB capacity. All three drives were mounted on a PC equipped with Intel® Core i7-3770 Processor (4 cores, 8 threads), 8GB memory, running Ubuntu 14.04 LTC. The PC acted as an in-house CSP server. Initially, the verifier was implemented on the same PC to minimize the network latency. Parallel data retrieval from multiple storage nodes was implemented using a PHP program with the multi-threading extension *pthreads*. We also implemented the verifier on a laptop equipped with an Intel® Core i7-6820HQ Processor (4 cores, 8 threads), 16GB memory running macOS Sierra 10.12.4. The laptop remotely challenged the server via an SSH connection, which was implemented using a PHP program with the secure communications library *phpseclib*. The code used for the experiments is available at [43].

Metrics: We evaluated our PoPR protocol for both correctness and soundness stated as follows.

Correctness: An honest CSP that stores a file $\mathbf{f}$ according to a pre-agreed layout should always pass the timing test.

Soundness: A dishonest CSP that stores a file $\mathbf{f}$ to fewer storage nodes should always violate the timing test.

To evaluate these two properties, we measured the response time of an honest and a dishonest CSP under various storage node configurations. The main metrics for the response time is its distribution over repeated challenges, and the gap between the response time and the deadlines set by the verifier. Before performing the timing test experiments, we profiled the storage nodes and the network to establish relevant bounds for the processing and network delays, so that the corresponding deadlines could be set accordingly.

6.1 Processing Delay Bounds

We first studied the processing delay $T^{(P)}$ for retrieving symbols of various sizes and computing their hash values. These experiments were performed to obtain the bounds of the processing delay for varying technologies. Figures 6(a), 6(b), and 6(c) show the processing delay of HDD_1, HDD_2, and SSD_1, respectively, as a function of the number of the randomly retrieved symbols. The variance is also shown. For HDD_1 and HDD_2, $T^{(P)}$ grows linearly to the number of retrieved symbols. This indicates a relatively constant processing rate, independently of the symbol size. For SSD_1, $T^{(P)}$ grows sublinearly for sizes less than 128KB and is almost linear for 256KB. The two HDDs have similar average processing rate, which varied about 10%.

To further understand the variation in $T^{(P)}$, we measured the average, minimum, and maximum delay ratio between HDD_1 and SSD_1. We omitted HDD_2 from our comparison, because its profile is similar to that of HDD_1. Figure 6(d) shows the processing delay ratio as a function of

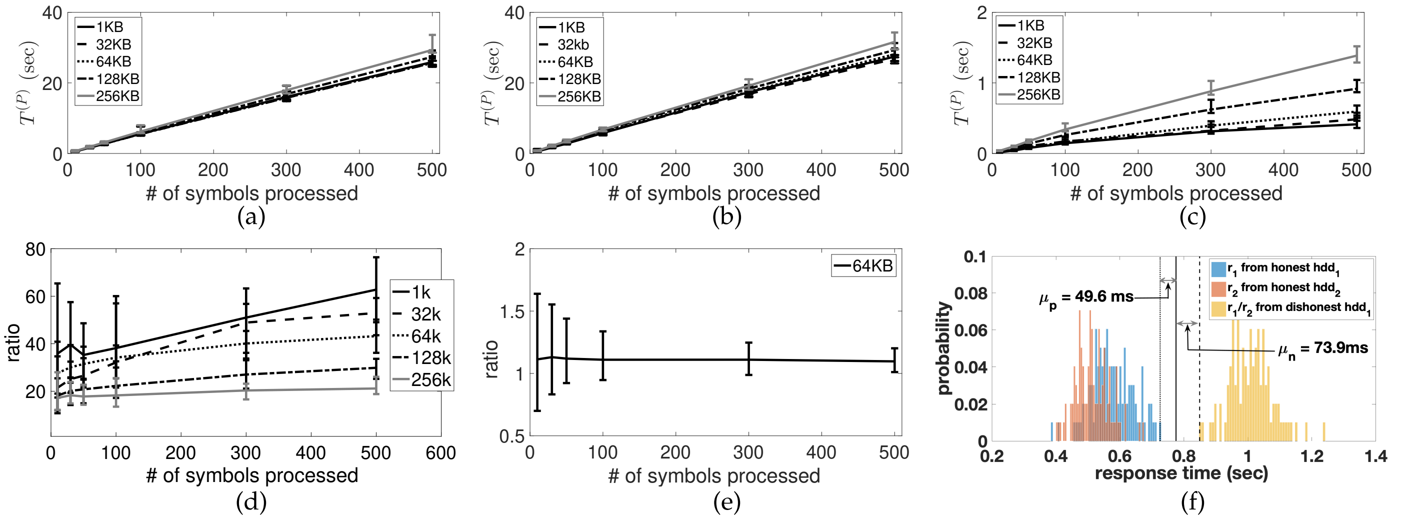

Fig. 6: (a) $T^{(P)}$ at HDD_1, (b) $T^{(P)}$ at HDD_2, (c) $T^{(P)}$ at SSD_1, (d) Processing delay ratio between SSD_1 and HDD_1, (e) Processing delay ratio between HDD_1 and HDD_2, (f) Probability distribution of t_1 and t_2 when storing $\mathbf{c}$ at two HDDs (honest CSP) and only HDD_1 (dishonest CSP).

the number of symbols. We observe that the ratio decreases with the symbol size and increases with the number of symbols retrieved. The processing delay of SSD_1 was up to 80 times smaller than that of HDD_1. The processing delay ratio between the two HDDs is less than two times with an average that is independent of the number of symbols retrieved, as shown in Fig. 6(e).

We use the results of Fig. 6(d) to select the bounds for the timing tests. The longest processing delay was 76.33ms (13.1 symbols/sec) whereas the smallest processing delay was 0.73ms (1,364 symbols/sec). Among the two HDDs, the recorded smallest processing delay was 46.33ms (21.58 symbols/sec). On the SSD, the recorded longest processing delay was 2.7ms (370.37 symbols/sec). For the experiments involving only HDDs, we set $T^{(P)}_{\min} = 45$ms (22 symbols/sec) and $T^{(P)}_{\max} = 85$ms (12 symbols/sec) . This allowed us to test the case where $T^{(P)}_{\max}/T^{(P)}_{\min} < 2\left(1 - (T^{(N)}_{\max}-T^{(N)}_{\min})/T^{(P)}_{\min}(y_i+y_j)\right)$. For the experiments involving both HDD and SSD, $T^{(P)}_{\min}$ was set to 0.73ms (1,368 symbols/sec). *Note that in a real scenario, the bounds could be set according to current specifications. The verifier does not have to probe the CSP to estimate the processing delay and the network latency.*

6.2 Network Latency Bounds

When the verifier was implemented on the same PC as the in-house CSP server, we measured the network latency by recording the round trip time (i.e., system delay) for sending a challenge and receiving a 256-bit hash value from the CSP. The hash was precomputed and stored at the CSP. The results of 1,000 repeated measurements are shown in Fig. 7(a). The network latency varied from $T^{(N)}_{\min} = 5.06$ms to $T^{(N)}_{\max} = 10.54$ms. We observe a considerable variation in the network latency, despite having the verifier on the same platform as the CSP. This latency is primarily attributed to how the operating system handles the request. Compared to the processing delay, the network latency in this scenario is equivalent to the retrieval of up to 14 symbols when the fastest SSD speed is considered.

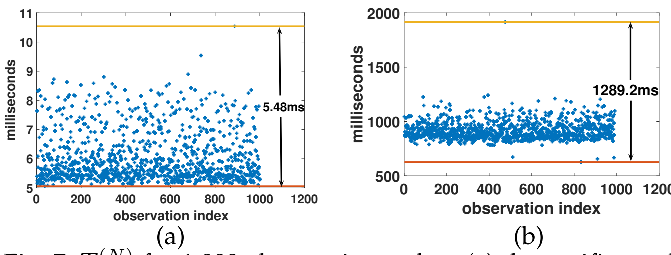

Fig. 7: $T^{(N)}$ for 1,000 observations when (a) the verifier and the CSP are implemented in the same machine and (b) the verifier uses SSH to the in-house CSP.

When the verifier was implemented on a laptop connected to the in-house CSP via an SSH connection, the network latency was recorded as the round trip time between sending the challenge Q and receiving a 256-bit hash over the SSH connection. The results of 1,000 measurements shown in Fig. 7(b), indicate a $T^{(N)}_{\min} = 626.3$ms and a $T^{(N)}_{\max} =$ 1,915.5ms. Here, the major component of the latency is the slowdown imposed by the SSH protocol.

6.3 Homogeneous Storage - Two HDDs

In this set of experiments, we considered the scenario where $\mathbf{c}$ is stored at HDD_1 and HDD_2. Although the same storage technology is used, the results in Section 6.1 verify that some variation exists in the processing delay between the two drives. For these experiments, the symbol size was set to 64KB and the verifier ran at the same host as the CSP. The network and processing delay bounds were set based on the results of Sections 6.1 and 6.2 to $T^{(N)}_{\min} = 5$ms, $T^{(N)}_{\max} =$ 10.6ms, $T^{(P)}_{\min} = 45$ms, and $T^{(P)}_{\max} = 85$ms. For these values, inequality (17) is satisfied for any positive y_1 and y_2, hence both inequalities in (11) can be simultaneously satisfied. The verifier set $y_1 = y_2 = 9$ symbols and also set the deadlines for r_1 and r_2 to $\tau(y_1) = \tau_(y_2) = y_1 T^{(P)}_{\max} + T^{(N)}_{\max} = 775.6$ms.

Figure 6(f) shows the distribution of the response time t_1 for retrieving 9 symbols from HDD_1, the response time t_2 for retrieving 9 symbols from HDD_2, and the response time for retrieving 18 symbols from HDD_1 only, across 100

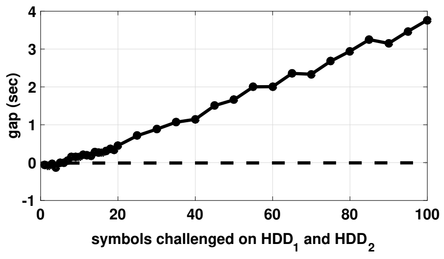

Fig. 8: Gap as a function of the # of the challenged symbols.

challenges. The latter case corresponds to a dishonest CSP who stores all symbols on HDD_1. The solid line represents the deadline of 775.6ms for any 9-symbol retrieval. The dashed line represents the fastest response recorded for any retrieval of 18 symbols on the dishonest HDD_1. Finally, the dotted line represents the slowest response recorded for any retrieval of 9 symbols on HDD_2. We observe that responses from the honest case always meet the deadline, indicating that our scheme achieves correctness. On the other hand, a dishonest CSP who stores data on a single storage node always misses the deadline, indicating that our scheme satisfies the soundness property.

The difference between the slowest response recorded from an honest CSP (dotted line) and the deadline (solid line) is 49.6ms. This difference is a measure of correctness and we call it the *false positive margin* μ_p. Moreover, the difference between the fastest response by a dishonest CSP (the dashed line) and the deadline (the solid line) is 73.9ms. This difference is a measure of soundness for the PoPR and we call it the *false negative margin* μ_n. As long as the deadline is set between the dotted and the dashed line, the correctness and soundness properties are guaranteed. We define the gap $\lambda = \mu_p + \mu_n$ to represent the overall margin for selecting the deadline. A larger λ implies a more robust scheme to any detection errors due to inaccuracies in the selection of the bounds or clock measurements. Figure 8 shows the gap, λ, as a function of the number of symbols challenged on each hard drive. We observe the gap grows linearly with the number of symbols challenged.

6.4 Heterogeneous Storage - One HDD and One SSD

In this set of experiments, the encoded file $\mathbf{c}$ was stored at one HDD_1 and one SSD_1. The symbol size was fixed to 64KB. The verifier set $T_{\min}^{(P)} = 0.73$ms and $T_{\max}^{(P)} = 85$ms. Due to the large difference between $T_{\min}^{(P)}$ and $T_{\max}^{(P)}$ no solution could simultaneously satisfy both inequalities in (11). We further considered two scenarios for the network latency. In the scenario where CSP and verifier resided on the same platform, the verifier set $T_{\min}^{(N)} = 5.0$ms, and $T_{\max}^{(N)} = 10.6$ms. The verifier chose to satisfy the second inequality in (11), by setting y_1 =1,200 symbols and $y_2 = 10$ symbols. The deadlines for r_1 and r_2 were set to $\tau(y_1) = y_1 T_{\max}^{(P)} + T_{\max}^{(N)} = 102$s and $\tau(y_2) = y_2 T_{\max}^{(P)} + T_{\max}^{(N)} = 0.86$s.

We first show that an honest CSP satisfies correctness by meeting the deadlines for the responses r_1 and r_2. Figure 9 shows the distribution of the response time for retrieving 10 symbols from HDD_1 in the honest case across 1,000 challenges. The response time for retrieving 1,200 symbols from SSD_1 is not shown because it is very small compared to the deadline (the deadline is 102s whereas the response time is less than one second). We note that the CSP always meets the deadline and passes the PoPR challenge.

We then considered dishonest CSP who stores all 1210 symbols (10 from r_1 and 1,200 from r_2) at the SSD drive which is the fastest. We retrieved 1,210 symbols from SSD_1 across 1,000 challenges and show the recorded response times in Fig. 9. We observe that when $\mathbf{x}_2$ is served after $\mathbf{x}_1$ the dishonest CSP always misses the deadline, indicating that the soundness property is guaranteed. Moreover, the false positive and false negative margins where $\mu_p = 94.1$ms and $\mu_n = 485.7$ms, respectively, showing an acceptable margin of operation for possible variations in the clock measurements.

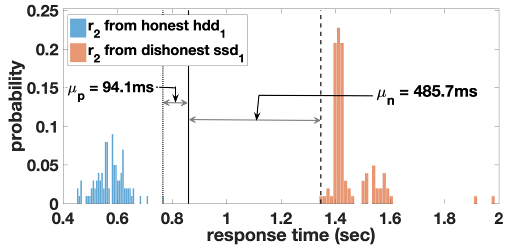

Fig. 9: PDF of t_2 when $\mathbf{c}$ is stored at HDD_1 and SSD_1 (honest CSP) and only SSD_1 (dishonest CSP) (symbol size 64KB, SSD: 1200 symbols, HDD: 10 symbols).

To demonstrate the superiority of our PoPR approach compared to prior works, we evaluated if a dishonest CSP is detected in the heterogeneous storage scenario for schemes relying on the fixed retrieval delay assumption. In our comparison, we considered the RAFT protocol because it is the only one that targets physical storage reliability without any computational constraints on the CSP. We set the symbol size to $64KB$ and considered two queries $\mathbf{x}_1$ and $\mathbf{x}_2$ constructed according to RAFT, with $y_1 = 100$ and $y_2 = 100$ symbols. The dishonest CSP stored all symbols at the SSD and first retrieved $\mathbf{x}_1$, followed by $\mathbf{x}_2$. The deadlines for each response were set according to the longest recorded delay on the HDDs as $\tau_1 = \tau_2 = y_1 \times T_{\max,HDD}^{P} = 100 \times 76.33ms = 7.633s$. Figure 10 shows the distribution of the response time for $\mathbf{x}_1$ and $\mathbf{x}_2$ over 100 challenges[2]. We observe that the dishonest CSP always meets the deadlines for both $\mathbf{x}_1$ and $\mathbf{x}_2$ by a wide margin (over seven seconds), indicating that the CSP can easily defeat the RAFT protocol, using a faster storage media than the one assumed.

Note that if the verifier treats the timing test of RAFT assuming that the SSD storage technology is used, an honest CSP will fail the test if it uses a slower media for storage. We repeated our previous experiment by storing $\mathbf{c}$ at one HDD and one SSD, set $y_1 = 100$ and $y_2 = 100$ as before, but setting the deadlines according to the recorded longest processing delay no the SSD as $\tau_1 = \tau_2 = y_1 \times T_{\max,SSD}^{P} = 100 \times 2.7ms = 0.27s$. Figure 11 shows the distribution of the response time for responses r_1 and r_2 over 100 challenges. We observe that although the response from the SSD consistently meets the deadline, the response from the

2. The time distributions for r_1 and r_2 are concentrated around 0.2sec and 0.3sec, respectively. They are shown as spikes because their variance is quite small relative to the deadline order of magnitude.

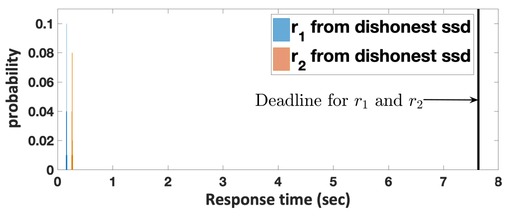

Fig. 10: PDF of responses r_1 and r_2 when all symbols are stored at the SSD, but deadlines are set according to an HDD model.

HDD always misses it, thus correctness is not achieved. This clearly demonstrates that RAFT is not technology-agnostic and the exact media where parts of $\mathbf{c}$ are distributed must be known for a successful test.

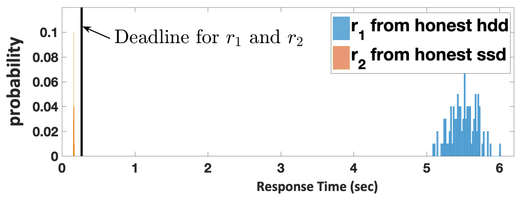

Fig. 11: PDF of responses r_1 and r_2 when all symbols are stored at the SSD, but deadlines are set according to an SDD model.

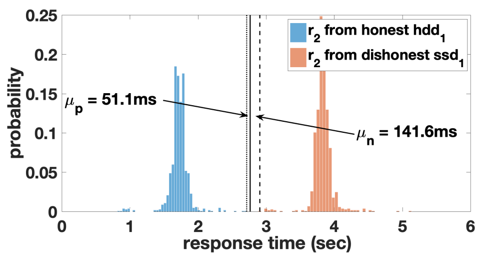

Fig. 12: Probability distribution of t_2 when storing $\mathbf{c}$ at HDD_1 and SSD_1 (honest CSP) and only SSD_1 (dishonest CSP) for network latency realized via an SSH connection (symbol size 64KB, SSD: 3000 symbols, HDD: 10 symbols).

We further evaluated how the verifier designs the challenge, when the network latency is significant. To add latency, the in-house CSP was remotely accessed via an SSH connection. The related network latency bounds were set to $T^{(N)}_{\min} = 0.6$s and $T^{(N)}_{\max} = 1.9$s. To satisfy the second inequality in (11), the length of the challenge to the first storage node had to be increased from 1,200 symbols to 3,000 symbols, while y_2 remained to be 10 symbols. Note that these values are quite large due to the extreme value of the network latency considered here (in the order of seconds). The respective deadlines were calculated by the verifier to $\tau(y_1) = y_1 T^{(P)}_{\max} + T^{(N)}_{\max} = 256.9$s and $\tau_{y_2} = y_2 T^{(P)}_{\max} + T^{(N)}_{\max} = 2.8$s.

Figure 12 shows the distribution of the response time for retrieving 10 symbols from HDD_1 in the honest case and the response time of retrieving 3,010 symbols from SSD_1 in the malicious case, across 1,000 challenges. The response time for retrieving 3,000 symbols from SSD_1 is not shown because it is very small compared to the deadline. We observe that when the CSP serves the long challenge first, it always misses the deadline for the short challenge. This choice occurs with probability 1/2 because the challenge lengths are hidden. Figure 13(a) shows the detection probability as a function of the number of challenge rounds α. As low as six challenges are sufficient to drive the detection probability close to one. We observed that when $\mathbf{x}_2$ is served after $\mathbf{x}_1$, r_2 from the honest case always meet the deadline, whereas that from the dishonest CSP misses the deadline. Moreover, the false positive and false negative margins are $\mu_p = 51.1$ms, and $\mu_n = 141.6$ms.

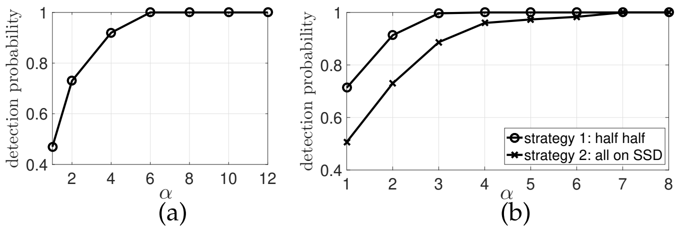

Fig. 13: Probability of detection as a function of the number of challenges/challenge rounds (α) for (a) two storage nodes, and (b) three storage nodes and different redistribution strategies.

6.5 Heterogeneous Storage - One SSD and Two HDDs

In the last set of experiments, we considered a CSP with three storage nodes. We deleted the data from the slowest HDD and implemented two data redistribution strategies. In strategy 1, the data from the deleted HDD was equally distributed to the remaining HDD and SSD, whereas in strategy 2, all data was stored at the SSD, which is the fastest drive. The processing delay and network latency bounds were set based on the results in Sections 6.2 and 6.1 to $T^{(P)}_{\min} = 0.73$ms and $T^{(P)}_{\max} = 85$ms, $T^{(N)}_{\min} = 5$ms and $T^{(N)}_{\max} = 10.6$ms.

Because of the bound difference and the geometric increase in the number of symbols to be challenged with the number of simultaneously challenged nodes, we opted to perform pairwise tests on (SN_1, SN_2), (SN_1, SN_3), and (SN_2, SN_3). For the first pair, we set $y_1 = 1,200$ and $y_2 = 10$, which satisfy (15). The deadlines for each pair were set accordingly to $\tau(y_1) = y_1 T^{(P)}_{\max} + T^{(N)}_{\max} = 102.01$s and $\tau_{y_2} = y_2 T^{(P)}_{\max} + T^{(N)}_{\max} = 0.86$s. The same deadlines were set for the remaining two pairs. The verifier cycled through challenging all three pair combinations and issued the respective challenges.

Figure 13(b) shows the detection rate as a function of the number of challenge rounds α. For either redistribution strategy, a few challenge rounds is sufficient to drive the detection probability close to one. Moreover, we observe that the detection probability under strategy 2 is always lower than strategy 1. This is intuitive because only one node stores deleted data in strategy 2. Redistributing the deleted data to a single node reduces the detection probability under pairwise tests, because the node violating the test is included only in two out of the three pairs.

7 PoPR Without Local Copies

In the basic PoPR protocol, the verifier maintains a local copy of $\mathbf{f}$ to perform the integrity test. However, when a large repository is outsourced, maintaining a local copy for verification purposes is costly. In this section, we modify the basic PoPR protocol to achieve the same security properties in terms of physical storage verification, without storing $\mathbf{f}$ at the verifier. The improved PoPR does not require the knowledge of $\mathbf{f}$ by the verifier, thus allowing the outsourcing of the verification service to a third party without compromising the client's privacy. These additional properties come at the inevitable expense in storage overhead for hosting the verifiable version of $\mathbf{f}$ at the CSP and increased communication cost for executing the challenge/response protocol.

Without a local copy of $\mathbf{f}$, the verifier faces two main challenges: (a) the integrity test on a response $\mathbf{r}_i$ to a challenge $\mathbf{x}_i$ cannot rely on the knowledge $\mathbf{f}$ and (b) the mechanism for hiding the length of $\mathbf{x}$ by retrieving symbols sequentially has to be redesigned. To tackle these two challenges, we present a solution that relies on the insertion of pseudo-random symbols called "verification symbols" that can be regenerated at the verifier from a single random seed. The idea of "spot-checking" verification symbols has been adopted in several PoR protocols (e.g., [4], [20], [23]), but our goal here is more challenging because the length of $\mathbf{x}$ must remain secret and the retrieved symbols must be revealed sequentially.

7.1 Setup Phase

During the setup phase, the client encodes $\mathbf{f}$ to a verifiable version $\mathbf{c}$. To generate $\mathbf{c}$, the client relies on a secret master key K, which is the only information stored for verification.

1) **Key generation.** Using the master key K of length τ, the client generates secret keys K_Π, K_{enc}, and K_{ver}.
2) **File partition.** The client divides $\mathbf{f}$ to n_s subfiles $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n_s}$. Each subfile is k symbols long.
3) **ECC application.** The client encodes each subfile $\mathbf{f}_i$ to $\mathbf{v}_i$ by applying an $(n-\gamma, k, d)_q$ ECC code $\mathcal{C}$. The encoded version of $\mathbf{f}$ is denoted by $\mathbf{v}$.
4) **Encryption.** The client encrypts each symbol in $\mathbf{v}_i$ with a semantically-secure encryption function E using key K_{enc}. The encrypted file is denoted by $\mathbf{m}$. The encryption is used to destroy the symbol dependency created by the application of the ECC and make parity and information symbols indistinguishable.
5) **Verification symbol generation and insertion.** For each subfile $\mathbf{m}_i$, the client applies a pseudo-random function $\phi : \{0,1\}^\tau \times [\![1,\gamma]\!] \rightarrow \{0,1\}^\ell$ to generate γ pseudo-random verification symbols $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_\gamma)$, where $w_i = \phi(K_{ver}, i)$. The symbols are appended to each $\mathbf{m}_i$ and create $\mathbf{m}'_i$. The subfile $\mathbf{m}'_i$ has a total of n symbols.
6) **Permutation.** The client creates $\mathbf{c}_i$ by applying pseudo-random permutation $\Pi_F : \{0,1\}^\tau \times [\![1,n]\!] \rightarrow [\![1,n]\!]$ to each $\mathbf{m}_i$. The permutation receives as input key K_Π and the subfile index i and is used to randomize the locations of the parity, information, and verification symbols.

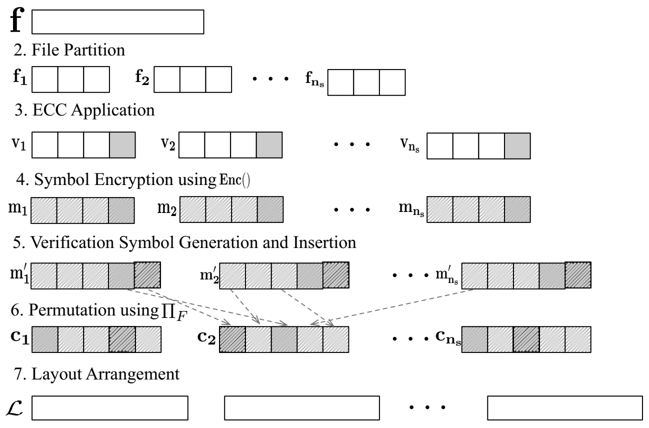

Fig. 14: Setup phase of the Improved PoPR.

7) **Layout arrangement.** A layout $\mathcal{L} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n\}$ is created by arranging $\mathbf{c}$ to n sectors, with each sector $\mathbf{s}_i$ hosting one symbol per $\mathbf{c}_i$.
8) **Distribution.** The client provides $\mathbf{c}$ and $\mathcal{L}$ to the CSP. If the verifier is implemented by a third party, the client provides K_Π, K_{enc}, and K_{ver} to the verifier.

The key differences between the setup of the basic PoPR and the PoPR without a local copy are steps 4, 5 and 6. In Step 4, each encoded subfile is encrypted with K_{enc} to destroy the symbol dependency between information and parity symbols. This is necessary because the CSP knows that the integrity of information and parity symbols cannot be checked by the verifier, as the latter no longer stores $\mathbf{f}$. To enable integrity verification with minimal storage cost, in step 5, pseudo-random verification symbols are generated and inserted to the file. The symbols are permuted using a pseudo-random permutation in Step 6. At step 6, information, parity, and verification symbols are indiscernible to the CSP and their locations within each sector remain unknown. The integrity test solely relies on the responses for the verification symbols alone. However, the timing test relies on the response time for all the symbols.

7.2 Verification Phase

In the verification phase, the verifier constructs a challenge Q which is sent to the CSP. The CSP must respond with a valid response R within the defined deadlines similar to our basic PoPR. However, constructing Q, R, and checking the integrity of R are more complicated due to the absence of a local copy $\mathbf{f}$ at the verifier.

Construction of the Challenge Q.

1) **Deadline selection:** The verifier specifies the number of symbols $y_1, y_2, \ldots, y_n$ to be queried from each storage node (sector). The verifier also selects n deadlines $\tau(y_1), \tau(y_2), \ldots, \tau(y_n)$. The deadlines are set according to the description in Section 4.5, similar to the basic PoPR scheme.
2) **Index permutation.** The verifier selects a random nonce η of length τ and applies a pseudo-random permutation $\Pi_Q : \{0,1\}^\tau \times [\![1,n_s]\!] \rightarrow [\![1,n_s]\!]$ to the indices of each sector $\mathbf{s}_i$. The permutation receives as input the nonce η and the sector index i, and creates a new arrangement on the indices of the

symbols in $\mathbf{s}_i$ denoted by $\Pi_Q(\eta, i)$. The nonce and the sector index randomize $\Pi_Q(\eta, i)$ per sector and per challenge construction.

3) **Terminating symbol selection.** The verifier selects a random nonce η' and n random verification symbols $c_{(a_{y_1,1}),1}$, $c_{(a_{y_2,2}),2}, \dots, c_{(a_{y_n,n}),n}$, one from each sector, where $(a_{j,i})$ is the index of the j^{th} symbol in $\mathbf{x}_i$. Verification symbol $c_{(a_{y_i,i}),i}$ denotes the last symbol in $\mathbf{x}_i$ for sector $\mathbf{s}_i$. The verifier computes $h(c_{(a_{y_1,1}),1}, \eta'), h(c_{(a_{y_2,2}),2}, \eta'), \dots, h(c_{(a_{y_n,n}),n}, \eta')$ for each terminating symbol, where $h(\cdot)$ is a public secure hash function.
4) **First symbol selection.** For each sector $\mathbf{s}_i$, the verifier traces the index of the $(y_i - 1)$th symbol preceding $c_{(a_{y_i,i}),i}$ in permutation $\Pi_Q(\eta, i)$. This becomes the first symbol $c_{(a_{1,i}),i}$ in $\mathbf{x}_i$ for sector $\mathbf{s}_i$.
5) **Challenge construction.** The challenge Q consists of the indices of the first symbol to be queried per sector $(a_{1,1}, a_{1,2} \dots, a_{1,n})$, the hash of the last symbol to be queried from each sector with η', namely $h(c_{(a_{y_1,1}),1}, \eta'), h(c_{(a_{y_2,2}),2}, \eta'), \dots, h(c_{(a_{y_n,n}),n}, \eta')$, η, and η'. The verifier sends Q to the CSP and initializes n clocks $t_1, t_2, \dots, t_n$ to zero. Π_Q is publicly known.

The PoPR without a local copy has some notable differences with the basic PoPR. First, the next symbol to be retrieved by the CSP is no longer a function of the previous one, as in Step 3 of Section 4.3, because $\mathbf{f}$ is not known at the verifier. To overcome this, the verifier applies a pseudo-random permutation on the symbol indices for each sector to create a random symbol order. Moreover, the symbols in a query $\mathbf{x}_i$ are selected in reverse. The verifier selects one verification symbol $c_{(a_{y_i,i}),i}$ at random to serve as the terminating symbol for the query. The $\mathbf{x}_i$ includes $y_i - 1$ symbols preceding $c_{(a_{y_i,i}),i}$ in the permutation (if the first symbol is reached, the verifier wraps around cyclically) plus the terminating symbol. The terminating symbol must be a verification symbol so that its hash value with η' can be computed at the verifier (verification symbols can be regenerated using K_{ver} and ϕ, whereas information symbols are no longer stored). The use of the hash of the last symbol forces the CSP to retrieve every symbol in $\mathbf{x}_i$ and hash it with η' to determine if the end of the query is reached. This property preserves the secrecy of the query length and also leads to serial symbol retrieval.

Figure 15 shows a toy example on the construction of $\mathbf{x}_1$ for a sector $\mathbf{s}_1 = \{c_{1,1}, \dots, c_{9,1}\}$ with nine symbols. Here, $c_{2,1}$ and $c_{6,1}$ are verification symbols and $y_1 = 3$.. The verifier first applies a pseudo-random permutation $\Pi_Q(\eta, 1)$ on the indices of $\mathbf{s}_1$, creating a new arrangement $\mathbf{s}'_1$. Next, the verifier picks verification symbol $c_{2,1}$ to be the terminating symbol in $\mathbf{x}_1$. The two symbols prior to $c_{2,1}$ on $\mathbf{s}'_1$ are included with the query resulting in $\mathbf{x}_1 = \{c_{5,1}, c_{1,1}, c_{2,1}\}$. In the challenge Q, the verifier includes the index of the first symbol in $\mathbf{x}_1$, which is $(5, 1)$, the hash of the last symbol $h(c_{2,1}, \eta')$, and nonces η and η'.

Construction of the Response R

1) **Permutation recovery.** The CSP computes $\Pi_Q(\eta, i)$ for each sector $\mathbf{s}_i$.

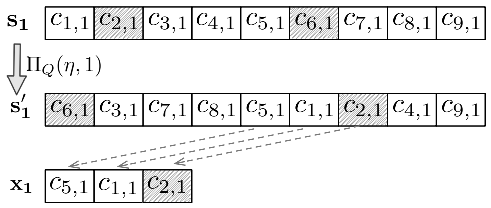

Fig. 15: Construction of challenge $\mathbf{x}_1$ for $\mathbf{s}_1$.

2) **Symbol retrieval.** The CSP retrieves the symbols in each $\mathbf{x}_i$ starting from $c_{(a_{1,i}),i}$ according to permutation $\Pi_Q(\eta, i)$. Symbols at different storage nodes are retrieved in parallel.
3) **Query end.** With every retrieved symbol $c_{(a_{j,i}),i}$, the CSP checks if $h(c_{(a_{j,i}),i}, \eta') \overset{?}{=} h(c_{(a_{y_i,i}),i}, \eta')$ to identify the query end.
4) Upon the retrieval of all symbols in $\mathbf{x}_i$ and their corresponding hash values the CSP computes response
$$\begin{aligned} r_i &= h(h(c_{(a_{1,i}),i}, \eta'), h(c_{(a_{2,i}),i}, \eta'), \\ &\quad \dots, h(c_{(a_{y_i,i}),i}, \eta')). \end{aligned}$$
The CSP sends r_i to the verifier as it becomes available, and then sends the individual hash values $h(c_{(a_{1,i}),i}, \eta'), h(c_{(a_{2,i}),i}, \eta'), \dots, h(c_{(a_{y_i,i}),i}, \eta')$ to the verifier.
5) **Verification.** The verifier records the arrival times of each response r_i and performs the timing test on each r_i. The verifier further collects the individual hash values and performs an integrity test. The CSP passes verification if both the timing and integrity tests are passed.

Following with our example in Fig. 15, to construct R, the CSP uses the public permutation Π_Q and η to obtain $\mathbf{s}'_1$. The CSP starts the symbol retrieval from $c_{5,1}$, hashing every retrieved symbol with η', until the hash matches $h(c_{2,1}, \eta')$. The CSP returns $r_1 = h(h(c_{5,1}, \eta'), h(c_{1,1}, \eta'), h(c_{2,1}, \eta'))$ to the verifier, followed by the individual hash values, $h(c_{5,1}, \eta'), h(c_{1,1}, \eta'), h(c_{2,1}, \eta')$.

Integrity Test. The integrity test checks the correctness of the response R returned by the CSP. This is necessary to ensure that the CSP indeed retrieved the intended symbols and did not create responses based on random symbol values. The integrity test is as follows:

1) **Response.** The verifier receives $R = \{r'_1, r'_2, \dots, r'_n\}$ from the CSP. For each r'_i, the verifier also receives the individual hashes $h(c'_{(a_{1,i}),i}, \eta')$, $h(c'_{(a_{2,i}),i}, \eta'), \dots, h(c'_{(a_{y_i,i}),i}, \eta')$ for each retrieved symbol.
2) **Response check.** The verifier first checks if
$$r'_i \overset{?}{=} h(h(c'_{(a_{1,i}),i}.\eta'), h(c'_{(a_{2,i}),i}, \eta'), \dots, h(c'_{(a_{y_i,i}),i}, \eta'))$$
to verify that the same symbols were used in the construction of r'_i and the individual hashes.
3) **Verification symbol generation.** The verifier regenerates all the verification symbols in each $\mathbf{x}_i$ by using ϕ, K_{ver}, Π_F, Π_Q, and the respective nonces.
4) **Verification symbol check.** The verifier checks if the has of each verification symbol with η' matches the individual hash returned by the CSP.

5) **Test success.** The CSP passes the integrity test if all returned values are correct in steps 1 and 4, and for all sectors.

For the example in Fig. 15, the verifier first checks if

$$r'_1 \overset{?}{=} h(h(c'_{5,1}.\eta'), h(c'_{1,1}, \eta'), h(c'_{2,1}, \eta')).$$

He then regenerates $c_{2,1}$ and checks if $h(c'_{2,1}, \eta') \overset{?}{=} h(c_{2,1}, \eta')$. If both checks are passed, the integrity test for x_1 is passed.

Timing test. The timing test verifies that the storage layout is not violated and the symbols are retrieved in parallel from multiple storage nodes. The steps of the timing test are the same as those of the basic PoPR, outlined in Section 4.5, and are not repeated here. The only difference is in that the symbols to be retrieved can be computed without having to retrieve each symbol. However, the CSP has to retrieve each symbol and compute its hash value with η' to find out whether the last symbol has been retrieved. Skipping the retrieval of several symbols in $\mathbf{x}_i$ is not helpful because the CSP may skip the terminating symbol and fail both the integrity (due to missing verification symbols) and timing (due to increasing the retrieval time) tests.

7.3 Security Analysis

Due to space limitations, we only provide a sketch of the security analysis for the PoPR without local copies. For the correctness property, it is straightforward to show that an honest CSP who maintains the agreed layout $\mathcal{L}$ will pass both the integrity and timing tests. For the integrity test, the CSP computes the correct hash values upon retrieval of the symbols. For the timing test, the deadlines are met because the symbols are retrieved in parallel according to $\mathcal{L}$.

For the security property, the analysis of the basic PoPR described in Section 5 (Proposition 2) holds when the CSP violates the agreed layout $\mathcal{L}$ by storing the symbols of the verifiable file version $\mathbf{c}$ to fewer storage nodes. If the CSP attempts to retrieve all the symbols indicated in Q to avoid failing the integrity test, the timing test will fail because of the cumulative processing delay in symbol retrieval. *However, the probabilistic nature of the integrity test gives rise to two new misbehavior strategies.* Without loss of generality, let the CSP be bound to fail the timing test for sector $\mathbf{s}_i$, because $\mathbf{s}_i$ stores the symbols of a deleted sector $\mathbf{s}_j$. In strategy 1, the CSP replies with a random r'_i to meet the deadline for $\mathbf{x}_i$ and then continues to retrieve all symbols in $\mathbf{x}_i$ to reply with the correct individual symbol hashes. To pass the response check in Step 2 of the integrity test in this case, the randomly selected r'_i must be equal to the hash of the individual hashes with η', which occurs with negligible probability when secure hashes are used (equal to a collision probability for a hash function).

In strategy 2, the CSP retrieves only a subset of the symbols in $\mathbf{x}_i$ to reduce the processing delay and meet the deadline for $\mathbf{x}_i$. For the retrieved symbols, the correct individual hash values $h(c_{(a_{j,i}),i}, \eta')$ are submitted and used in the computation of r_i, whereas random values are used for the symbols that were not retrieved. Under strategy 2, the CSP will pass the response check of Step 2 because the hash of all the hashes will match with the returned r'_i. The security of the protocol depends on the omission of verification symbols from the response. If the hash of one verification symbol is omitted, a violation is detected during the verification symbol check (Step 4) of the integrity test. The CSP's success probability here can be made arbitrarily small by inserting enough verification symbols at the expense of additional storage overhead at the CSP.

Note that the use of a verification symbol as a terminating symbol reveals the role of that symbol as a verification symbol. When many challenges are executed, a subset of verification symbols can be tagged by the CSP. The tagged verification symbols will not be omitted under strategy 2 to increase the CSP's chances in passing the integrity test. To address this problem, the verifier can limit the terminating symbols to a subset of verification symbols. We call these terminating verification symbols as *guards*. Therefore, there will always be verification symbols that remain unknown to the CSP and hence lead to detection of integrity violations. We emphasize that knowing the location of guards does not reveal the length of each challenge, because the guards are hashed with a fresh nonce with every challenge. Therefore the identity and location of the guard is hidden until the guard is retrieved.

File confidentiality. A desirable property of our PoPR construction is that the auditing process can be outsourced to a third-party verifier while preserving the confidentiality of the outsourced file. As the integrity and timing tests rely only on the use of the verification symbols, knowledge of K is sufficient to construct challenges and perform the tests. The information symbols in $\mathbf{f}$ need not be known at the verifier and are not revealed by the responses of the CSP (the information symbols are hashed with η' in each response).

8 Conclusion and Open Problems

We developed two Proof of Physical Reliability (PoPR) auditing mechanisms which prove that a CSP stores an outsourced repository across multiple physical storage nodes. Our mechanism relies on a combination of storage integrity and timing tests to verify the parallel symbol retrieval from multiple storage nodes. The basic PoPR construction requires the storage of a local file copy at the verifier to perform the physical storage verification and comes at no storage overhead to the CSP. We further proposed a PoPR construction for large repositories that does not require any storage at the verifier beyond a single cryptographic key. The later scheme comes at the expense of additional storage for the CSP. Compared with the state-of-the-art, we showed analytically and via experimentation that our approach accommodates CSPs with heterogeneous storage devices (hard disks, SSDs, etc.) and does not require constant data access nor network delays. Instead, it can be configured to operate under any delay variance, because it relies only on (loose) delay bounds.

The present work does not jointly consider the requirements for concurrently proving the logical and physical reliability. A PoR and a PoPR test each independently impose the retrieval of a subset of symbols that are being outsourced to the CSP. One possible extension of this work is to consider the joint design of PoR and PoPR tests, such as that the two can be executed under a single setup and interactive verification phase. Moreover, our PoPR construction relies on a pre-agreed storage layout. Construction a PoPR scheme for heterogeneous storage nodes that does not require agreement to an a prior layout remains an open challenge. Finally, the use of loose bounds on the processing and network delays comes at the expense of higher communication overhead for performing PoPR audits. A verifiable processing and network delay estimation method could reduce this overhead without jeopardizing security.

Acknowledgments

The authors thank the anonymous reviewers for their insightful comments. This research was supported in part by the US National Science Foundation under grant CNS1813401. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of the US National Science Foundation.

References

[1] Nasuni, "The state of cloud storage: 2015 industry report," http://www6.nasuni.com/rs/445-ZDB-645/images/Nasuni-White-Paper-2015-State-of-Cloud-Storage.pdf, 2015.
[2] Forrester, "Study: Cloud service agreements omit key considerations new ISO/IEC 19086-1 standard guides organizations to structured, effective agreements," http://download.microsoft.com/download/7/7/E/77E57C7E-4458-47A7-8646-8AA6F2BC7EED/Cloud_Service_Agreements_Omit_Key_Considerations-Forrester_Paper.pdf, 2016.
[3] M. Krigsman, "The Linkup: When the cloud fails," http://www.zdnet.com/blog/projectfailures/mediamax-the-linkup-when-the-cloud-fails/999.
[4] A. Juels and B. S. Kaliski Jr, "PORs: Proofs of retrievability for large files," in *Proc. of the 14th ACM CCS Conference*, 2007, pp. 584–597.
[5] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2013.
[6] Z. Hao, S. Zhong, and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE Trans. on Knowledge and Data Engineering*, vol. 23, no. 9, pp. 1432–1437, 2011.
[7] 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.
[8] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. of the 16th ACM CCS Conference*, 2009, pp. 213–222.
[9] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. of the 4th Securecom Conference*, 2008, pp. 1–9.
[10] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession," in *Proc. of the ICDCS Conference*, 2008, pp. 411–420.
[11] B. Chen, A. K. Ammula, and R. Curtmola, "Towards server-side repair for erasure coding-based distributed storage systems," in *Proc. of the 5th ACM Conference on Data and Application Security and Privacy*, 2015, pp. 281–288.
[12] Z. Wang, K. Sun, S. Jajodia, and J. Jing, "Disk storage isolation and verification in cloud," in *Proc. of the IEEE GLOBECOM Conference*, 2012, pp. 771–776.
[13] Z. Wang, K. Sun, J. Jing, and S. Jajodia, "Verification of data redundancy in cloud storage," in *Proc. of the 2013 International Workshop on Security in Cloud Computing*, 2013, pp. 11–18.
[14] A. Albeshri, C. Boyd, and J. G. Nieto, "Enhanced GeoProof: improved geographic assurance for data in the cloud," *International Journal of Information Security*, vol. 13, no. 2, pp. 191–198, 2014.
[15] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest, "How to tell if your cloud files are vulnerable to drive crashes," in *Proc. of the 18th ACM CCS Conference*, 2011, pp. 501–514.
[16] M. Gondree and Z. N. Peterson, "Geolocation of data in the cloud," in *Proc. of the Third ACM Conference on Data and Application Security and Privacy*, 2013, pp. 25–36.
[17] K. Benson, R. Dowsley, and H. Shacham, "Do you know where your cloud files are?" in *Proc. of the 3rd ACM Workshop on Cloud Computing Security*, 2011, pp. 73–82.
[18] G. J. Watson, R. Safavi-Naini, M. Alimomeni, M. E. Locasto, and S. Narayan, "LoSt: Location based storage," in *Proc. of the 2012 ACM Workshop on Cloud Computing Security*, 2012, pp. 59–70.
[19] B. Chen and R. Curtmola, "Towards self-repairing replication-based storage systems using untrusted clouds," in *Proc. of the third ACM conference on Data and application security and privacy*, 2013, pp. 377–388.
[20] K. D. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2009, pp. 187–198.
[21] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 220–232, 2012.
[22] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, "Remote data checking using provable data possession," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 12, 2011.
[23] M. Azraoui, K. Elkhiyaoui, R. Molva, and M. Önen, "StealthGuard: Proofs of retrievability with hidden watchdogs," in *ESORICS 2014, 19th European Symposium on Research in Computer Security, September 7-11, 2014, Wroclaw, Poland*, Wroclaw, POLAND, 2014.
[24] D. Vasilopoulos, K. Elkhiyaoui, R. Molva, and M. Onen, "POROS: proof of data reliability for outsourced storage," in *Proceedings of the 6th International Workshop on Security in Cloud Computing*, ser. SCC '18, 2018, pp. 27–37.
[25] F. Armknecht, L. Barman, J.-M. Bohli, and G. O. Karame, "Mirror: Enabling proofs of data replication and retrievability in the cloud," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 1051–1068.
[26] B. Fisch, "Tight proofs of space and replication," in *Advances in Cryptology – EUROCRYPT 2019*, 2019, pp. 324–348.
[27] O. M. Vasilopoulos, Dimitrios and R. Molva, "Proof of data reliability for real-world distributed outsourced storage," in *to appear in the Proc. of the International Conference on Security and Cryptography*, 2019.
[28] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," in *Proc. of the 14th ACM CCS Conference*, 2007, pp. 598–609.
[29] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of Retrievability via Hardness Amplification," in *Proc. of the 6th Theory of Cryptography Conference on Theory of Cryptography (TTC 2009)*, 2009, pp. 109–127.
[30] H. Shacham and B. Waters, "Compact Proofs of Retrievability," in *Proc. of the Asiacrypt Conference*, 2008, pp. 90–107.
[31] J. Li, X. Tan, X. Chen, D. S. Wong, and F. Xhafa, "OPoR: Enabling proof of retrievability in cloud computing with resource-constrained devices," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 195–205, April 2015.
[32] Y. Li, A. Fu, Y. Yu, and G. Zhang, "IPOR: An efficient ida-based proof of retrievability scheme for cloud storage systems," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.
[33] A. Fu, Y. Li, S. Yu, Y. Yu, and G. Zhang, "DIPOR: an ida-based dynamic proof of retrievability scheme for cloud storage systems," *Journal of Network and Computer Applications*, vol. 104, pp. 97 – 106, 2018.
[34] M. B. Paterson, D. R. Stinson, and J. Upadhyay, "Multi-prover proof of retrievability," *Journal of Mathematical Cryptology*, vol. 12, no. 4, pp. 203–220, 2018.
[35] F. Armknecht, J.-M. Bohli, G. O. Karame, Z. Liu, and C. A. Reuter, "Outsourcing proofs of retrievability," in *to appear in the IEEE Transactions on Cloud Computing*, 2018.
[36] C. B. Tan, M. H. A. Hijazi, Y. Lim, and A. Gani, "A survey on proof of retrievability for cloud data integrity and availability: Cloud storage state-of-the-art, issues, solutions and future trends," *Journal of Network and Computer Applications*, vol. 110, pp. 75–86, 2018.
[37] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," in *Proc. of the IEEE 17th International*

Symposium on High Performance Computer Architecture, 2011, pp. 266–277.

[38] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard, "A cooperative internet backup scheme," in *Proc. of the USENIX Conference*, 2003, pp. 3–27.

[39] T. S. Schwarz and E. L. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," in *Proc. of the 26th IEEE International Conference on Distributed Computing Systems*, 2006, pp. 12–12.

[40] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure Coding in Windows Azure Storage," in *Procedings of the USENIX ATC Conference*, June 2012.

[41] M. Sathiamoorthy, M. Asteris, D. S. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing Elephants: Novel Erasure Codes for Big Data," *CoRR*, vol. abs/1301.3791, 2013.

[42] M. N. Krishnan, N. Prakash, V. Lalitha, B. Sasidharan, P. V. Kumar, S. Narayanamurthy, R. Kumar, and S. Nandi, "Evaluation of Codes with Inherent Double Replication for Hadoop," in *Proc. Usenix HotStorage*, vol. abs/1406.6783, Philadelphia, PA, Jun. 2014.

[43] L. Li, "Github repository," https://github.com/liliarizona/PoPR_paper, 2018.

Li Li is a PhD candidate in Electrical and Computer Engineering at the University of Arizona. She obtained a M.S. degree in Electrical and Computer Engineering from the University of Arizona and a B.E. degree in Electronics Information Engineering from Huazhong University of Science and Technology, China. Her research interests are in the areas of proofs of data integrity in cloud storage systems.

Loukas Lazos is an Associate Professor of Electrical and Computer Engineering at the University of Arizona. He received his Ph.D. in Electrical Engineering from the University of Washington in 2006. In 2007, he was the co-director of the Network Security Lab at the University of Washington. Dr. Lazos joined the the University of Arizona in August 2007. His main research interests are in the areas of network security, user privacy, wireless communications, network performance analysis, and network visualization. He is a recipient of the NSF CAREER Award (2009), for his research in security of multi-channel wireless networks. He was the general co-chair for the ACM WiSec 2012 Conference and served as the TPC co-chair for the IEEE CNS 2018 Conference, the Communication and Information System Security Symposium at GLOBECOM 2013, and the 4th IEEE International Workshop on Data Security and Privacy in Wireless Networks (DSPAN) 2013. He has served on organizing and technical program committees of numerous international conferences and on panels for several funding agencies.