PORTOS: Proof of Data Reliability for Real-World Distributed Outsourced Storage

Dimitrios Vasilopoulos, Melek Önen and Refik Molva
EURECOM, Sophia Antipolis, France

Keywords: Secure Cloud Storage, Proofs of Reliability, Reliable Storage, Verifiable Storage.

Abstract: Proofs of data reliability are cryptographic protocols that provide assurance to a user that a cloud storage system correctly stores her data and has provisioned sufficient redundancy to be able to guarantee reliable storage service. In this paper, we consider distributed cloud storage systems that make use of erasure codes to guarantee data reliability. We propose a novel proof of data reliability scheme, named PORTOS, that on the one hand guarantees the retrieval of the outsourced data in their entirety through the use of proofs of data possession and on the other hand ensures the actual storage of redundancy. PORTOS makes sure that redundancy is stored at rest and not computed on-the-fly (whenever requested) thanks to the use of time-lock puzzles. Furthermore, PORTOS delegates the burden of generating the redundancy to the cloud. The repair operations are also taken care of by the cloud. Hence, PORTOS is compatible with the current cloud computing model where the cloud autonomously performs all maintenance operations without any interaction with the user. The security of the solution is proved in the face of a *rational* adversary whereby the cheating cloud provider tries to gain storage savings without increasing its total operational cost.

1 INTRODUCTION

Distributed storage systems guarantee data reliability by redundantly storing data on multiple storage nodes. Depending on the redundancy mechanism, each storage node may store either a full copy of the data (replication) or a fragment of an encoded version of it (erasure codes). In the context of a trusted environment such as an on-premise storage system, this type of setting is sufficient to guarantee data reliability against accidental data loss caused by random hardware failures or software bugs.

With the prevalence of cloud computing, outsourcing data storage has become a lucrative option for many enterprises as they can meet their rapidly changing storage requirements without having to make tremendous upfront investments for infrastructure and maintenance. However, in the cloud setting, outsourcing raises new security concerns with respect to misbehaving or malicious cloud providers. An economically motivated cloud storage provider could opt to "take shortcuts" when it comes to data maintenance in order to maximize its returns and thus jeopardizing the reliability of user data. In addition, cloud storage providers currently do not assume any liability in case data is lost or damaged. Hence, cloud users may be reluctant to adopt cloud storage services for applications that are otherwise perfectly suited for the cloud such as archival storage or backups. As a result, technical solutions that aim at allowing users to verify the long-term integrity and reliability of their data, would be beneficial both to customers and providers of cloud storage services.

Focusing on the long term integrity requirement, the literature features a number of solutions for verifiable outsourced storage. Notably, proofs of retrievability (PoR) (Juels and Kaliski, 2007; Shacham, H. and Waters, B., 2008) and data possession (PDP) (Ateniese et al., 2007) are cryptographic mechanisms that enable a cloud user to efficiently verify that her data is correctly stored by an untrusted cloud storage provider.

PoRs and PDPs have been extended in order to ensure data reliability over time introducing the notion of proofs of data reliability (Curtmola et al., 2008; Bowers et al., 2009; Armknecht et al., 2016; Vasilopoulos et al., 2018). In addition to the integrity of a user's outsourced data, a proof of data reliability provides the user with the assurance that the cloud storage provider has provisioned sufficient redundancy to be able to guarantee reliable storage service. In a straightforward approach, the cloud user

Vasilopoulos, D., Önen, M. and Molva, R.
PORTOS: Proof of Data Reliability for Real-World Distributed Outsourced Storage.
DOI: 10.5220/0007927301730186
In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE 2019)*, pages 173-186
ISBN: 978-989-758-378-0

locally generates the necessary redundancy and subsequently stores the data together with the redundancy on multiple storage nodes. Previous work has established that, in the case of erasure code-based storage systems, the relation between the data and redundancy symbols should stay hidden from the cloud storage provider, otherwise, the latter could store a portion of the encoded data only and compute any missing symbols upon request. Similarly, in the case of replication based storage systems, each storage node should store a different replica; otherwise, the cloud storage provider could simply store a single replica. As a result, most proof of data reliability schemes (Curtmola et al., 2008; Chen and Curtmola, 2017; Bowers et al., 2009; Chen et al., 2015) require some interaction with the user to repair damaged data. Hence these schemes are at odds with automatic maintenance that is a key feature of cloud storage systems.

In this paper, we propose PORTOS, a proof of data reliability scheme tailored to distributed cloud storage systems that makes use of erasure codes to guarantee data reliability. PORTOS achieves the following properties against a rational cloud storage provider:

- *Data reliability against t storage node failures:* PORTOS uses a systematic linear erasure code to add redundancy to the outsourced data which thereafter stores across multiple storage nodes. The system can tolerate the failure of up to t storage nodes, and successfully reconstruct the original data using the contents of the surviving ones.
- *Data and redundancy integrity:* PORTOS leverages a PDP scheme to verify the integrity of the outsourced data and it further takes advantage of the homomorphic properties of the PDP tags, in order to verify the integrity of the redundancy. Moreover, thanks to the combination of the PDP scheme with erasure codes, PORTOS can provide a cloud user with the assurance that she can recover her data in their entirety.
- *Automatic data maintenance by the cloud storage provider:* In PORTOS, the cloud storage provider has the means to generate the required redundancy, detect storage node failures and repair corrupted data entirely on its own without any interaction with the user conforming to the current cloud model. This setting, however, allows a malicious cloud storage provider to delete a portion of the encoded data and compute any missing symbols upon request. To defend against such an attack, PORTOS relies on time-lock puzzles, and masks the data, making the symbol regeneration process time-consuming. In this way, a rational cloud is provided with a strong incentive to conform to the proof of data reliability protocol.
- *Real-world cloud storage architecture:* PORTOS conforms to the current model of erasure-code-based distributed storage systems. Moreover, it does not make any assumption regarding the system's underlying technology as opposed to prior proof of data reliability schemes that also allow for automatic data maintenance by the cloud storage provider.

In summary, we make the following contributions in this paper:

- We propose a new formal definition for proofs of data reliability, which is more generic than the definitions presented in prior work (Armknecht et al., 2016).
- We present a novel proof of data reliability scheme, named PORTOS, that on the one hand guarantees that the outsourced data can be retrieved in their entirety through the use of proofs of data possession and on the other hand ensures the actual storage of redundancy. PORTOS makes sure that redundancy is stored at rest and not computed on-the-fly (whenever requested) thanks to the use of time-lock puzzles. Furthermore, PORTOS delegates the burden of generating the redundancy, as well as the repair operations, to the cloud storage provider.
- We show that PORTOS is secure against a rational adversary, and we further evaluate the impact of two types of attacks, considering in both cases the most favorable scenario for the adversary. Finally, we evaluate the performance of PORTOS.

The remaining of this paper is organized as follows: In Section 2, we give an overview of prior work in the field. We describe the formal definition, adversary model, and security requirements of a proof of data reliability scheme in Section 3. In Section 4, we introduce PORTOS, a novel data reliability scheme and we analyze its security in Section 5. Finally, in Section 6 we analyze the performance of PORTOS.

2 PRIOR WORK

Replication-based Proofs of Data Reliability. The authors in (Curtmola et al., 2008) propose a proof of data possession scheme (PDP) which extends the PDP scheme in (Ateniese et al., 2007) and enables the client to verify that the cloud provider stores at least t replicas of her data. In (Chen and Curtmola, 2013; Chen and Curtmola, 2017), the replica differentiation

mechanism in (Curtmola et al., 2008) is replaced by a tunable masking mechanism, which allows to shift the bulk of repair operations to the cloud storage provider with the user acting as a repair coordinator. In (Leontiadis and Curtmola, 2018), the scheme in (Chen and Curtmola, 2017) is extended in order to construct a proof of data reliability protocol that allows for cross-user file-level deduplication. The authors in (Barsoum and Hasan, 2012; Barsoum and Hasan, 2015) propose a multi-replica dynamic PDP scheme that enables clients to update/insert selected data blocks and to verify multiple replicas of their outsourced files. The scheme in (Etemad and Küpçü, 2013), extends the dynamic PDP scheme in (Erway et al., 2009) in order to transparently support replication in distributed cloud storage systems. In (Armknecht et al., 2016), the authors propose a multi-replica PoR scheme that delegates the replica construction to the cloud storage provider. The scheme uses tunable puzzles as part of its replication mechanism in order to force the cloud storage provider to store the replicas at rest.

Erasure-code-based Proofs of Data Reliability. The authors in (Bowers et al., 2009) propose HAIL which provides a high availability and integrity layer for cloud storage. HAIL uses erasure codes in order to guarantee data retrievability and reliability among distributed storage servers, and enables a user to detect and repair data corruption. The work in (Chen et al., 2015) redesigns parts of (Bowers et al., 2009) in order to achieve a more efficient repair phase that shifts the bulk computations to the cloud side. In (Chen et al., 2010), the authors present a remote data checking scheme for network-coding-based distributed storage systems that minimizes the communication overhead of the repair component compared to erasure coding-based approaches. The work in (Le and Markopoulou, 2012) extends the scheme in (Chen et al., 2010) by improving the repair mechanism in order to reduce the computation cost for the client, and introducing a third party auditor. Based on the introduction of this new entity, the authors in (Thao and Omote, 2016) design a network-coding-based PoR scheme in which the repair mechanism is executed between the cloud provider and the third party auditor without any interaction with the client. The authors in (Bowers et al., 2011) propose RAFT, an erasure-code-based protocol that can be seen as a proof of fault tolerance. RAFT relies on technical characteristics of rotational hard drives in order to construct a time-based challenge, that enables a client to verify that her encoded data is stored at multiple storage nodes within the same data center. In (Vasilopoulos et al., 2018), the authors propose POROS, a proof of data reliability scheme for erasure-code-based cloud storage systems which combines PDP with time constrained operations in order to force the cloud storage provider to store the redundancy at rest. POROS dictates that all the redundancy is stored permuted on a single storage node and leverages the technical characteristics of rotational hard drives to set a time threshold for the generation of the proof. Both schemes enable the outsourcing of the data encoding to the cloud storage provider as well as automatic maintenance operations without any interaction with the client.

Most of the proof of data reliability schemes presented above share a common system model, where the user generates the required redundancy locally, before uploading it together with the data to the cloud storage provider. Furthermore, when corruption is detected, the cloud cannot repair it autonomously, because either it expects some input from another entity or all computations are performed by the client. The solution in (Armknecht et al., 2016) outsources to the cloud the redundancy generation, however, it relies on replication to provide data reliability and hence is not directly comparable to our proof of data reliability scheme.

PORTOS is directly comparable with the work in (Bowers et al., 2011) and in (Vasilopoulos et al., 2018), in the sense that it also uses erasure codes to guarantee data reliability and it delegates the burden of generating the redundancy, as well as the repair operations, to the cloud storage provider. Nonetheless, in contrast to these schemes, our solution does not make any assumption regarding the underlying storage technology. Namely, both schemes rely on technical characteristics of rotational hard drives in order to set a time-threshold for the cloud servers to respond to a read request for a set of data blocks. Furthermore, unlike in PORTOS, the challenge verification in (Bowers et al., 2011) requires that a copy of the encoded data is stored locally at the user. Lastly, the proof of data reliability scheme in (Vasilopoulos et al., 2018) requires that all redundancy symbols are stored the result on a single storage node without fragmentation. On the contrary, PORTOS does not deviate from the standard model of erasure-code based distributed storage systems.

3 PROOFS OF DATA RELIABILITY

3.1 Environment

We consider a setting where a user U produces a data object $\mathcal{D}$ from a file D and subsequently outsources $\mathcal{D}$ to an untrusted cloud storage provider C who commits to store $\mathcal{D}$ in its entirety across a set of n storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$ with *reliability guarantee* t: storage service guarantee against t storage node failures. We define a *proof of data reliability* scheme as a protocol executed between the cloud storage provider C with its storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$ on the one hand and a verifier V on the other hand. The aim of such a protocol is to enable V to check and validate (i) the *integrity* of $\mathcal{D}$ and, (ii) whether the *reliability guarantee* t is satisfied. In order not to cancel out the storage and performance advantages of the cloud, all this verification should be performed without V downloading the entire content associated to $\mathcal{D}$ from $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$. We consider two different settings for a proof of data reliability scheme: a private one where U is the verifier and a public one where V can be any third party except for C.

Formal Definition. A *proof of data reliability* scheme comprises seven polynomial time algorithms:

$\mathsf{Setup}\ (1^{\lambda},t)\rightarrow(\{\mathcal{S}^{(j)}\}_{1\leq j\leq n},\mathsf{param}_{\mathsf{system}})$: This algorithm takes as input the security parameter λ and the reliability parameter t, and returns the set of storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$, the system parameters $\mathsf{param}_{\mathsf{system}}$, and the specification of the redundancy mechanism.

$\mathsf{Store}\ (1^{\lambda},D)\rightarrow(\mathrm{K}_{\mathsf{U}},\mathcal{D},\mathsf{param}_{\mathcal{D}})$: This randomized algorithm invoked by the user U takes as input the security parameter λ and the to-be-outsourced file D, and outputs the user key K_{U}, the verifiable data object $\mathcal{D}$, which also includes a unique identifier fid, and optionally, a set of data object parameters $\mathsf{param}_{\mathcal{D}}$. In a private proof of data reliability scheme $\mathrm{K}_{\mathsf{U}}:=\mathrm{sk}$ is the user's secret key, whereas in a public one $\mathrm{K}_{\mathsf{U}}:=(\mathrm{sk},\mathrm{pk})$ is the user's private/public key pair.

$\mathsf{GenR}\ (\mathcal{D},\mathsf{param}_{\mathsf{system}},\mathsf{param}_{\mathcal{D}})\rightarrow(\tilde{\mathcal{D}})$: This algorithm takes as input the verifiable data object $\mathcal{D}$, the system parameters $\mathsf{param}_{\mathsf{system}}$, and optionally, the data object parameters $\mathsf{param}_{\mathcal{D}}$, and outputs the data object $\tilde{\mathcal{D}}$. Algorithm GenR may be invoked either by the user U, when the user generates the redundancy on her own; or by C, when the redundancy computation is entirely outsourced to the cloud storage provider. Depending on the redundancy mechanism, $\tilde{\mathcal{D}}$ may comprise multiple copies of $\mathcal{D}$ or an encoded version of it. Additionally, algorithm GenR generates the necessary integrity values that will further help for the integrity verification of $\tilde{\mathcal{D}}$'s redundancy.

$\mathsf{Chall}\ (\mathrm{K}_{\mathsf{V}},\mathsf{param}_{\mathsf{system}})\rightarrow(\mathsf{chal})$: This stateful and probabilistic algorithm invoked by the verifier V takes as input the verifier key K_{V} and the system parameters $\mathsf{param}_{\mathsf{system}}$, and outputs a challenge chal. In a private proof of data reliability scheme the verifier key is the user's secret key ($\mathrm{K}_{\mathsf{V}}:=\mathrm{sk}$), whereas in a public one the verifier key is the user's public key ($\mathrm{K}_{\mathsf{V}}:=\mathrm{pk}$).

$\mathsf{Prove}\ (\mathsf{chal},\tilde{\mathcal{D}})\rightarrow(\mathsf{proof})$: This algorithm invoked by C takes as input the challenge chal and the data object $\tilde{\mathcal{D}}$, and returns C's response proof of data reliability.

$\mathsf{Verify}\ (\mathrm{K}_{\mathsf{V}},\mathsf{chal},\mathsf{proof},\mathsf{param}_{\mathcal{D}})\rightarrow(\mathsf{dec})$: This deterministic algorithm invoked by V takes as input C's proof corresponding to a challenge chal, the verifier key K_{V}, and optionally, the data object parameters $\mathsf{param}_{\mathcal{D}}$, and outputs a decision $\mathsf{dec}\in\{\mathsf{accept},\mathsf{reject}\}$ indicating a successful or failed verification of the proof, respectively. In a private proof of data reliability scheme the verifier key is the user's secret key ($\mathrm{K}_{\mathsf{V}}:=\mathrm{sk}$), whereas in a public one the verifier key is the user's public key ($\mathrm{K}_{\mathsf{V}}:=\mathrm{pk}$).

$\mathsf{Repair}\ ({}^{*}\tilde{\mathcal{D}},\mathsf{param}_{\mathsf{system}},\mathsf{param}_{\mathcal{D}})\rightarrow(\tilde{\mathcal{D}})$: This algorithm takes as input a corrupted data object ${}^{*}\tilde{\mathcal{D}}$ together with its parameters $\mathsf{param}_{\mathcal{D}}$ and the system parameters $\mathsf{param}_{\mathsf{system}}$, and either reconstructs $\tilde{\mathcal{D}}$ or outputs a failure symbol $\perp$. Algorithm Repair may be invoked either by U or C depending on the proof of data reliability scheme.

3.2 Adversary Model

Similar to (Armknecht et al., 2016; Vasilopoulos et al., 2018), we consider an adversary model whereby the cloud storage provider C is *rational*, in the sense that C decides to cheat only if it achieves some cost savings. For a proof of data reliability scheme that deals with the storage of data and its redundancy, a rational adversary would try to save some storage space without increasing its overall operational cost. The overall operational cost is restricted to the maximum number n of storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$ whereby each of them has a bounded capacity of storage and computational resources. More specifically, assume that for some proof of data reliability scheme there exists an attack which allows C to produce a

valid proof of data reliability while not fulfilling the reliability guarantee t. If in order to mount this attack, C has to provision either more storage resources or excessive computational resources compared to the resources required when it implements the protocol in a correct manner, then a *rational* C will choose not to launch his attack.

3.3 Security Requirements

A proof of data reliability scheme oughts to fulfill the following requirements.

Req 0 : **Correctness.** A proof of data reliability scheme should be correct: if both the cloud storage provider C and the verifier V are *honest*, then on input chal sent by the verifierV, using algorithm Chall, algorithm Prove (invoked by C) generates a Proof of Data Reliability proof such that algorithm Verify yields accept with probability 1. Put differently, an honest C should always be able to pass the verification of proof of data reliability.

Req 1 : **Extractability.** It is essential for any proof of data reliability scheme to ensure that an honest user U can recover her file D with high probability. This guarantee is formalized using the notion of extractability introduced in (Juels and Kaliski, 2007; Shacham, H. and Waters, B., 2008). If a cloud storage provider C can convince an honest verifier V with high probability that it is storing the data object $\mathcal{D}$ together with its respective redundancy, then there exists an extractor algorithm that given sufficient interaction with C can extract the file D. To define this requirement, we consider a game between an adversary $\mathcal{A}$ and an environment, where $\mathcal{A}$ plays the role of the prover likewise in (Shacham, H. and Waters, B., 2008). The environment simulates all honest users and verifiers, and it further provides $\mathcal{A}$ with oracles for the algorithms Setup , Store , Chall , and Verify . $\mathcal{A}$ interacts with the environment and requests O_{Store} to compute the tuple $(\mathrm{sk}, \mathcal{D}, \mathsf{param}_{\mathcal{D}})$ for several chosen files D and for different honest users. Thereafter, $\mathcal{A}$ invokes a series of proof of data reliability executions by interacting with O_{Chall} and O_{Verify}. Finally, $\mathcal{A}$ picks a user U and a tuple $(\mathrm{sk}, \mathcal{D}, \mathsf{param}_{\mathcal{D}})$, corresponding to some file D and simulates a cheating cloud storage provider C'. Let C' succeed in making algorithm Verify yield dec := accept in an non-negligible ε fraction of proof of data reliability executions. We say that the proof of data reliability scheme meets the extractability guarantee, if there exists an extractor algorithm such that given sufficient interactions with C', it recovers D.

Req 2 : **Soundness of Redundancy Generation.** In addition to the extractability guarantee a proof of data reliability scheme should ensure the soundness of the redundancy generation mechanism. This entails that, in the face of data corruption, the original file D can be effectively reconstructed using the generated redundancy. Hence, in proof of data reliability schemes wherein algorithm GenR is implemented by the cloud storage provider C, it is crucial to ensure that the latter performs this operation in a correct manner. Namely, an encoded data object should either consist of actual codeword symbols or all replicas should be genuine copies of the data object. In other words, the only way C can produce a valid proof of data reliability is by correctly generating the redundancy.

Req 3 : **Storage Allocation Commitment.** A crucial aspect of a Proof of Data Reliability scheme, is forcing a cloud storage provider C to store *at rest* the outsourced data object $\mathcal{D}$ *together with* the relevant redundancy. This requirement is formalized similarly to the storage allocation guarantee introduced in (Armknecht et al., 2016). A cheating cloud storage provider C' that participates in the above mentioned extractability game (see *Req 1*), and dedicates only a fraction of the storage space required for storing *both* $\mathcal{D}$ and its redundancy in their entirety, cannot convince the verifier V to accept its proof with overwhelming probability.

4 PORTOS

In this section, we present PORTOS: a proof of data reliability scheme. PORTOS is based on the use of erasure codes to offer reliable storage with automatic maintenance. A user U sends a data object $\mathcal{D}$ to a cloud storage provider C, which in turn encodes $\mathcal{D}$ using a systematic linear (k,n)-MDS code (Xing and Ling, 2003), and thereupon stores it across a set of n storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$ with reliability guarantee against t storage node failures. Moreover, in PORTOS, C has the means to generate the required redundancy, detect storage node failures, and repair corrupted data, entirely on its own, without any interaction with U.

Such setting, however, presents a cheating C with the opportunity to misbehave, for instance, by not fulfilling the storage allocation commitment requirement and computing the redundancy symbols on-the-fly upon request. To prevent such behavior, PORTOS leverages time-lock puzzles in order to augment the resources (storage and computational) a cheating C has to provision in order to produce a valid proof of

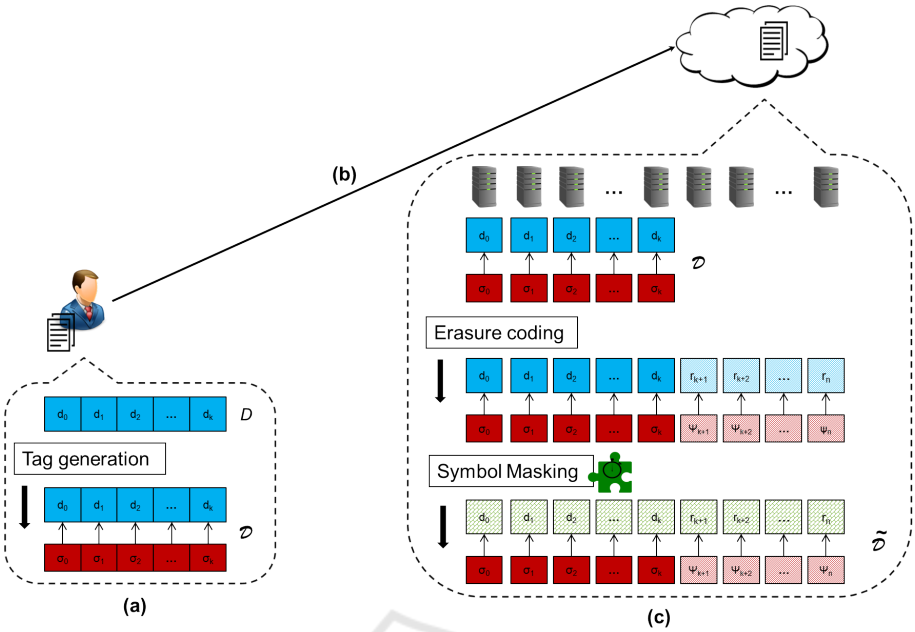

Figure 1: Overview of PORTOS outsourcing process: (a) The user U computes the linearly-homomorphic tags for the original data symbols; (b) U outsources the data object $\mathcal{D}$ to cloud storage provider C; (c) Using $\mathbf{G}$, C applies the systematic erasure code on both data symbols and their tags yielding the redundancy symbols and their corresponding tags; thereafter, C derives the masking coefficients and masks all data and redundancy symbols.

data reliability. Nonetheless, this mechanism does not incur any additional storage or computational cost to an honest C that generates the same proof. Moreover, the puzzle difficulty can be adapted to C's computational capacity as it evolves over time. Therefore, a *rational* C is provided with a strong incentive to conform to the proof of data reliability protocol rather than attempt to misbehave. Figure 1 depicts the steps of PORTOS outsourcing process.

4.1 Building Blocks

MDS Codes. *Maximum distance separable* (MDS) codes (Xing and Ling, 2003; Suh and Ramchandran, 2011) are a class of linear block erasure codes used in reliable storage systems that achieve the highest error-correcting capabilities for the amount of storage space dedicated to redundancy. A (k,n)-MDS code encodes a data segment of size k symbols into a codeword comprising n code symbols. The input data symbols and the corresponding code symbols are elements of a finite field $\mathbb{F}_q$, where q is a large prime and $k \leq n \leq q$. In the event of data corruption, the original data segment can be reconstructed from any set of k code symbols. Furthermore, up to $n-k+1$ corrupted symbols can be repaired. The new code symbols can either be identical to the lost ones, in which case we have exact repair, or can be functionally equivalent, in which case we have functional repair where the original code properties are preserved.

A systematic linear MDS-code has a generator matrix of the form $\mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}]$ and a parity check matrix of the form $\mathbf{H} = [-\mathbf{P}^\top \mid \mathbf{I}_{n-k}]$. Hence, in a systematic code, the code symbols of a codeword include the data symbols of the original segment. Reed-Solomon codes (Xing and Ling, 2003) are a typical example of MDS codes, their generator matrix $\mathbf{G}$ can be easily defined for any given values of (k,n), and are used by a number of storage systems (Blaum et al., 1994).

Linearly–homomorphic Tags. PORTOS's integrity guarantee derives from the use of linearly-homomorphic tags proposed, for the design of the private PoR scheme in (Shacham, H. and Waters, B., 2008).

This scheme consists of the following algorithms:

$\mathsf{SW.Store}\ (1^\lambda, D) \rightarrow (\mathsf{fid}, \mathsf{sk}, \mathcal{D})$: This randomized algorithm invoked by the user U, first picks a pseudo-random function $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \mathbb{Z}_q$, together with its pseudo-randomly generated key $\mathsf{k_{prf}} \in \{0,1\}^\lambda$ and a non-zero element $\alpha \xleftarrow{R} \mathbb{Z}_q$, and subsequently computes a homomorphic authentication tag for each symbol d_i of the file D as follows:

$$\sigma_i = \alpha d_i + \mathsf{PRF}(\mathsf{k_{prf}}, i) \in \mathbb{Z}_q, \quad \text{for } 1 \leq i \leq n.$$

Algorithm SW.Store then picks a unique identifier fid, and terminates its execution by outsourcing to the cloud storage provider C the authenticated data object:

$$\mathcal{D} := \{\mathsf{fid};\ \{d_i\}_{1\le i\le n};\ \{\sigma_i\}_{1\le i\le n}\}.$$

SW.Chall $(\mathsf{fid}, \mathsf{sk}) \rightarrow (\mathsf{chal})$: This algorithm invoked by U picks l random elements $\nu_c \in \mathbb{Z}_q$ and l random symbol indices i_c, and sends to C the challenge

$$\mathsf{chal} := \{(i_c, \nu_c)\}_{1\le c\le l}.$$

SW.Prove $(\mathcal{D}, \mathsf{chal}) \rightarrow (\mathsf{proof})$: Upon receiving the challenge chal, C invokes this algorithm which computes the proof $\mathsf{proof} = (\mu, \tau)$ as follows:

$$\mu = \sum_{(i_c,\nu_c)\in\mathsf{chal}} \nu_c\, d_{i_c}\ ,\quad \tau = \sum_{(i_c,\nu_c)\in\mathsf{chal}} \nu_c\, \sigma_{i_c}.$$

SW.Verify $(\mathsf{sk}, \mathsf{proof}, \mathsf{chal}) \rightarrow (\mathsf{dec})$: This algorithm invoked by U, verifies that the following equation holds:

$$\tau \stackrel{?}{=} \alpha\mu + \sum_{(i_c,\nu_c)\in\mathsf{chal}} \nu_c\, \mathsf{PRF}(\mathsf{k_{prf}}, i_c).$$

If proof is well formed, algorithm SW.Verify outputs dec := accept; otherwise it returns dec := reject.

Thanks to the unforgeability of homomorphic tags, a malicious C cannot corrupt outsourced data whilst eluding detection.

Time-lock Puzzles. A time-lock puzzle is a cryptographic function that requires the execution of a predetermined number of sequential exponentiation computations before yielding its output. The RSA-based puzzle of (Rivest et al., 1996) requires the repeated squaring of a given value β modulo N, where $N := p'q'$ is a publicly known RSA modulus, p' and q' are two safe primes[1] that remain secret, and $\mathcal{T}$ is the number of squarings required to solve the puzzle, which can be adapted to the solver's capacity of squarings modulo N per second. Thereby, $\mathcal{T}$ defines the puzzle's difficulty. Without the knowledge of the secret factors p' and q', there is no faster way of solving the puzzle than to begin with the value β and perform $\mathcal{T}$ squarings sequentially. On the contrary, an entity that knows p' and q', can efficiently solve the puzzle by first computing the value $e := 2^{\mathcal{T}}\ (mod\ \phi(N))$ and subsequently computing $\beta^e\ (mod\ N)$.

[1] such that 2 is guaranteed to have a large order modulo $\phi(N)$ where $\phi(N) = (p'-1)(q'-1)$

PORTOS leverages the cryptographic puzzle of (Rivest et al., 1996) to build a mechanism that enables a user U to increase the computational load of a misbehaving cloud storage provider C. To this end, C is required to generate a set of pseudo-random values, called masking coefficients, which are combined with the symbols of the encoded data object $\mathcal{D}$. C is expected to store at rest the masked data. More specifically, in the context of algorithm Store , U outputs two functions: the function maskGen which is sent to C together with $\mathcal{D}$ and is used by algorithms GenR and Repair ; and the function maskGenFast which is used by U within the scope of algorithm Verify .

maskGen$((i,j), \mathsf{param_m}) \rightarrow m_i^{(j)}$: This function takes as input the indices $(i,\ j)$, and the tuple $\mathsf{param_m} := (N, \mathcal{T}, \mathsf{PRG_{mask}}, \eta_\mathsf{m})$ comprising the RSA modulus $N := p'q'$, the squaring coefficient $\mathcal{T}$, a pseudo-random generator $\mathsf{PRG_{mask}} : \mathbb{Z}_N \times \{0,1\}^* \rightarrow \mathbb{Z}_N$ [2], and a seed $\eta_\mathsf{m} \in \mathbb{Z}_N$.

Function maskGen computes the masking coefficient $m_i^{(j)}$ as follows:

$$m_i^{(j)} := \left(\mathsf{PRG_{mask}}(\eta_\mathsf{m}, i \parallel j)\right)^{2^{\mathcal{T}}}\ (mod\ N).$$

maskGenFast$((i,j), (p',q'), \mathsf{param_m}) \rightarrow m_i^{(j)}$: In addition to $(i,\ j)$ and $\mathsf{param_m} := (N, \mathcal{T}, \mathsf{PRG_{mask}}, \eta_\mathsf{m})$, this function takes as input the secret factors $(p',\ q')$. Knowing p' and q', function maskGenFast efficiently computes the masking coefficient $m_i^{(j)}$ by first computing the value e:

$$\phi(N) := (p'-1)(q'-1),\ \ e := 2^{\mathcal{T}}\ (mod\ \phi(N)),$$
$$m_i^{(j)} := \left(\mathsf{PRG_{mask}}(\eta_\mathsf{m}, i \parallel j)\right)^{e}\ (mod\ N).$$

The puzzle difficulty can be adapted to the computational capacity of C as it evolves over time such that the evaluation of the function maskGen requires a noticeable amount of time to yield $m_i^{(j)}$. Furthermore, the masking coefficients are at least as large as the respective symbols of $\mathcal{D}$, hence storing the coefficients, as a method to deviate from our data reliability protocol, would demand additional storage resources which is at odds with C's primary objective.

4.2 Protocol Specification

We now describe in detail the algorithms of PORTOS.

Setup $(1^\lambda, t) \rightarrow (\{\mathcal{S}^{(j)}\}_{1\le j\le n}, \mathsf{param_{system}})$: Algorithm Setup first picks a prime number q, whose size

[2] such that its output is guaranteed to have a large order modulo N.

is chosen according to the security parameter λ. Afterwards, given the reliability parameter t, algorithm Setup yields the generator matrix $\mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}]$ of a systematic linear (k,n)-MDS code in $\mathbb{Z}_q$, for $k < n < q$ and $t \leq n-k+1$. In addition, algorithm Setup chooses n storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$ that are going to store the encoded data: the first k of them are data nodes that will hold the actual data symbols, whereas the rest $n-k$ are considered as redundancy nodes.

Algorithm Setup then terminates its execution by returning the storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$ and the system parameters $\mathsf{param}_{\mathsf{system}} := (k, n, \mathbf{G}, q)$.

U.Store $(1^\lambda, D, \mathsf{param}_{\mathsf{system}}) \rightarrow (\mathrm{sk}, \mathcal{D}, \mathsf{param}_{\mathcal{D}}$, maskGenFast): On input security parameter λ, file $D \in \{0,1\}^*$ and, system parameters $\mathsf{param}_{\mathsf{system}}$, this randomized algorithm first splits D into s segments, each composed of k data symbols. Hence D comprises $s \cdot k$ symbols in total. A data symbol is an element of $\mathbb{Z}_q$ and is denoted by $d_i^{(j)}$ for $1 \leq i \leq s$ and $1 \leq j \leq k$.

Algorithm U.Store also picks a pseudo-random function $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \mathbb{Z}_q$, together with its pseudo-randomly generated key $\mathsf{k}_{\mathsf{prf}} \in \{0,1\}^\lambda$, and a non-zero element $\alpha \xleftarrow{R} \mathbb{Z}_q$. Hereafter, U.Store computes for each data symbol a *linearly homomorphic* MAC as follows:

$$\sigma_i^{(j)} = \alpha d_i^{(j)} + \mathsf{PRF}(\mathsf{k}_{\mathsf{prf}}, i \parallel j) \in \mathbb{Z}_q.$$

In addition, algorithm U.Store produces a time-lock puzzle by generating an RSA modulus $N := p'q'$, where p' and, q' are two randomly-chosen safe primes of size λ bits each, and specifies the puzzle difficulty coefficient $\mathcal{T}$, and the time threshold $\mathsf{T}_{\mathsf{thr}}$. Thereafter, algorithm U.Store picks a pseudo-random generator $\mathsf{PRG}_{\mathsf{mask}} : \mathbb{Z}_N \times \{0,1\}^* \rightarrow \mathbb{Z}_N$[3] together with a random seed $\eta_{\mathsf{m}} \xleftarrow{R} \mathbb{Z}_N$, and constructs the functions maskGen and maskGenFast as described in Section 4.1

$$\mathsf{maskGen}\big((i,j),\ \mathsf{param}_{\mathcal{D}}\big) \rightarrow m_i^{(j)},$$
$$\mathsf{maskGenFast}\big((i,j),\ (p',q'),\ \mathsf{param}_{\mathcal{D}}\big) \rightarrow m_i^{(j)}.$$

Finally, algorithm U.Store picks a pseudo-random generator $\mathsf{PRG}_{\mathsf{chal}} : \{0,1\}^* \rightarrow [1,s]^l$ and a unique identifier fid.

Algorithm U.Store then terminates its execution by returning the user key

$$\mathrm{sk} := \big(\mathsf{fid},\ (\alpha, \mathsf{k}_{\mathsf{prf}}),\ (p',q',\mathsf{maskGenFast})\big),$$

[3] such that its output is guaranteed to have a large order modulo N.

Table 1: Notation used in the description of PORTOS.

Notation	Description
D	File to-be-outsourced
$\mathcal{D}$	Outsourced data object ($\mathcal{D}$ consists of data and PDP tags)
$\tilde{\mathcal{D}}$	Encoded and masked data object
$\mathcal{S}$	Storage node
$\mathbf{G}$	Generator matrix of the (n,k)-MDS code
$\alpha, \mathsf{k}_{\mathsf{prf}}$	Secret key used by the linearly homomorphic tags
j	Storage node index, $1 \leq j \leq n$, (there are n $\mathcal{S}$s in total)
i	Data segment index, $1 \leq i \leq s$, ($\mathcal{D}$ consist of s segments)
$d_i^{(j)}$	Data symbol, $1 \leq j \leq k$ and $1 \leq i \leq s$
$\tilde{d}_i^{(j)}$	Masked data symbol, $1 \leq j \leq k$ and $1 \leq i \leq s$
$\sigma_i^{(j)}$	Data symbol tag, $1 \leq j \leq k$ and $1 \leq i \leq s$
$r_i^{(j)}$	Redundancy symbol, $k+1 \leq j \leq n$ and $1 \leq i \leq s$
$\tilde{r}_i^{(j)}$	Masked redundancy symbol, $k+1 \leq j \leq n$ and $1 \leq i \leq s$
$\psi_i^{(j)}$	Redundancy symbol tag, $k+1 \leq j \leq n$ and $1 \leq i \leq s$
$m_i^{(j)}$	Masking coefficient, $1 \leq j \leq n$ and $1 \leq i \leq s$
η_{m}	Random seed used to generate $m_i^{(j)}$
p', q'	Primes for RSA modulus $N := p'q'$ of the time-lock puzzle
$\mathcal{T}$	Time-lock puzzle's difficulty coefficient
l	Size of the challenge
$\mathsf{T}_{\mathsf{thr}}$	Time threshold for the proof generation
$i_c^{(j)}$	Indices of challenged symbols, $1 \leq j \leq n$ and $1 \leq c \leq l$
$\eta^{(j)}$	Random seed used to generate $i_c^{(j)}$, $1 \leq j \leq n$
ν_c	Challenge coefficients, $1 \leq c \leq l$
$\tilde{\mu}^{(j)}$	Aggregated data/redundancy symbols, $1 \leq j \leq n$
$\tau^{(j)}$	Aggregated data/redundancy tags, $1 \leq j \leq n$
J_{f}	Set of failed storage nodes
J_{r}	Set of surviving storage nodes

the to-be-outsourced data object together with the integrity tags

$$\mathcal{D} := \Big\{\mathsf{fid};\ \{d_i^{(j)}\}_{\substack{1\leq j\leq k\\ 1\leq i\leq s}};\ \{\sigma_i^{(j)}\}_{\substack{1\leq j\leq k\\ 1\leq i\leq s}}\Big\},$$

and the data object parameters

$$\mathsf{param}_{\mathcal{D}} := \begin{pmatrix} \mathsf{PRG}_{\mathsf{chal}}, \mathsf{maskGen}, \\ \mathsf{param}_{\mathsf{m}} := (N, \mathcal{T}, \eta_{\mathsf{m}}, \mathsf{PRG}_{\mathsf{mask}}) \end{pmatrix}.$$

C.GenR $(\mathcal{D}, \mathsf{param}_{\mathsf{system}}, \mathsf{param}_{\mathcal{D}}) \rightarrow (\tilde{\mathcal{D}})$: Upon reception of data object $\mathcal{D}$, algorithm C.GenR starts computing the redundancy symbols $\{r_i^{(j)}\}_{\substack{k+1\leq j\leq n\\ 1\leq i\leq s}}$ by multiplying each segment $\mathbf{d}_i := \big(d_i^{(1)}, \ldots, d_i^{(k)}\big)$ with the generator matrix $\mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}]$:

$$\mathbf{d}_i \cdot [\mathbf{I}_k \mid \mathbf{P}] = \big(d_i^{(1)}, \ldots, d_i^{(k)} \mid r_i^{(k+1)}, \ldots, r_i^{(n)}\big).$$

Similarly, algorithm C.GenR multiplies the vector of linearly-homomorphic tags $\boldsymbol{\sigma}_i := \big(\sigma_i^{(1)}, \ldots, \sigma_i^{(k)}\big)$ with $\mathbf{G}$:

$$\boldsymbol{\sigma}_i \cdot [\mathbf{I}_k \mid \mathbf{P}] = \big(\sigma_i^{(1)}, \ldots, \sigma_i^{(k)} \mid \psi_i^{(k+1)}, \ldots, \psi_i^{(n)}\big).$$

One can easily show that $\{\psi_i^{(j)}\}_{\substack{k+1\leq j\leq n\\ 1\leq i\leq s}}$ are the *linearly-homomorphic* authenticators of $\{r_i^{(j)}\}_{\substack{k+1\leq j\leq n\\ 1\leq i\leq s}}$.

Thereafter, algorithm C.GenR generates the masking coefficients using the function maskGen:

$$\{m_i^{(j)}\}_{\substack{1\le j\le n\\ 1\le i\le s}} := \mathsf{maskGen}\big((i,j),\ \mathsf{param_m}\big)\ (mod\ q),$$

and then, masks all data and redundancy symbols as follows:

$$\{\tilde{d}_i^{(j)}\}_{\substack{1\le j\le k\\ 1\le i\le s}} \leftarrow \{d_i^{(j)} + m_i^{(j)}\}_{\substack{1\le j\le k\\ 1\le i\le s}}, \tag{1}$$

$$\{\tilde{r}_i^{(j)}\}_{\substack{k+1\le j\le n\\ 1\le i\le s}} \leftarrow \{r_i^{(j)} + m_i^{(j)}\}_{\substack{k+1\le j\le n\\ 1\le i\le s}}. \tag{2}$$

At this point, algorithm C.GenR deletes all masking coefficients $\{m_i^{(j)}\}_{\substack{1\le j\le n\\ 1\le i\le s}}$ and terminates its execution by returning the encoded data object

$$\tilde{\mathcal{D}} := \left\{ \mathsf{fid}\ ;\quad \begin{pmatrix} \{\tilde{d}_i^{(j)}\}_{\substack{1\le j\le k\\ 1\le i\le s}} \mid \{\tilde{r}_i^{(j)}\}_{\substack{k+1\le j\le n\\ 1\le i\le s}} \end{pmatrix} \atop \begin{pmatrix} \{\sigma_i^{(j)}\}_{\substack{1\le j\le k\\ 1\le i\le s}} \mid \{\psi_i^{(j)}\}_{\substack{k+1\le j\le n\\ 1\le i\le s}} \end{pmatrix} \right\},$$

and by storing the data symbols $\{\tilde{d}_i^{(j)}\}_{\substack{1\le j\le k\\ 1\le i\le s}}$ together with $\{\sigma_i^{(j)}\}_{\substack{1\le j\le k\\ 1\le i\le s}}$ and the redundancy symbols $\{\tilde{r}_i^{(j)}\}_{\substack{k+1\le j\le n\\ 1\le i\le s}}$ together with $\{\psi_i^{(j)}\}_{\substack{k+1\le j\le n\\ 1\le i\le s}}$ at the corresponding storage nodes.

U.Chall $(\mathsf{fid}, \mathsf{sk}, \mathsf{param_{system}}) \to (\mathsf{chal})$: Provided with the object identifier fid, the secret key sk, and the system parameters $\mathsf{param_{system}}$, algorithm U.Chall generates a vector $(\nu_c)_{c=1}^{l}$ of l random elements in $\mathbb{Z}_q$ together with a vector of n random seeds $(\eta^{(j)})_{j=1}^{n}$, and then, terminates by sending to all storage nodes $\{\mathcal{S}^{(j)}\}_{1\le j\le n}$ the challenge

$$\mathsf{chal} := \big(\mathsf{fid}, \big((\eta^{(j)})_{j=1}^{n},\ (\nu_c)_{c=1}^{l}\big)\big).$$

C.Prove $(\mathsf{chal}, \tilde{\mathcal{D}}, \mathsf{param}_{\mathcal{D}}) \to (\mathsf{proof})$: On input of challenge $\mathsf{chal} := \big(\mathsf{fid}, \big((\eta^{(j)})_{j=1}^{n}, (\nu_c)_{c=1}^{l}\big)\big)$, object parameters $\mathsf{param}_{\mathcal{D}} := (\mathsf{PRG_{chal}}, \mathsf{maskGen}, \mathsf{param_m})$, and data object $\mathcal{D}$ each storage node $\{\mathcal{S}^{(j)}\}_{1\le j\le n}$ invokes an instance of this algorithm and computes the response tuple $(\tilde{\mu}^{(j)}, \tau^{(j)})$ as follows:

It first derives the indices of the requested symbols and their respective tags

$$(i_c^{(j)})_{c=1}^{l} := \mathsf{PRG_{chal}}(\eta^{(j)}), \quad \text{for } 1 \le j \le n,$$

and subsequently, it computes the following linear combination

$$\tilde{\mu}^{(j)} \leftarrow \begin{cases} \sum_{c=1}^{l} \nu_c\ \tilde{d}_{i_c^{(j)}}^{(j)}, & \text{if } 1 \le j \le k \\ \sum_{c=1}^{l} \nu_c\ \tilde{r}_{i_c^{(j)}}^{(j)}, & \text{if } k+1 \le j \le n, \end{cases} \tag{3}$$

$$\tau^{(j)} \leftarrow \begin{cases} \sum_{c=1}^{l} \nu_c\ \sigma_{i_c^{(j)}}^{(j)}, & \text{if } 1 \le j \le k \\ \sum_{c=1}^{l} \nu_c\ \psi_{i_c^{(j)}}^{(j)}, & \text{if } k+1 \le j \le n. \end{cases} \tag{4}$$

Algorithm C.Prove terminates its execution by returning the set of tuples:

$$\mathsf{proof} := \big\{(\tilde{\mu}^{(j)}, \tau^{(j)})\big\}_{1\le j\le n}.$$

U.Verify $(\mathsf{sk}, \mathsf{chal}, \mathsf{proof}, \mathsf{maskGenFast}, \mathsf{param}_{\mathcal{D}}) \to (\mathsf{dec})$: On input of user key $\mathsf{sk} := (\alpha, \mathsf{k_{prf}}, p', q')$, challenge $\mathsf{chal} := \big(\mathsf{fid}, \big((\eta^{(j)})_{j=1}^{n}, (\nu_c)_{c=1}^{l}\big)\big)$, proof $\mathsf{proof} := \big\{(\tilde{\mu}^{(j)}, \tau^{(j)})\big\}_{1\le j\le n}$, function maskGenFast, and data object parameters $\mathsf{param}_{\mathcal{D}} := (\mathsf{PRG_{chal}}, \mathsf{maskGen}, \mathsf{param_m})$, this algorithm first checks if the response time of all storage nodes $\{\mathcal{S}^{(j)}\}_{1\le j\le n}$ is shorter than the time threshold $\mathsf{T_{thr}}$. If not algorithm U.Verify terminates by outputting $\mathsf{dec} := \mathsf{reject}$; otherwise it continues its execution and checks that all tuples $(\tilde{\mu}^{(j)}, \tau^{(j)})$ in proof are well formed as follows:

It first derives the indices of the requested symbols and their respective tags

$$(i_c^{(j)})_{c=1}^{l} := \mathsf{PRG_{chal}}(\eta^{(j)}), \quad \text{for } 1 \le j \le n,$$

and it generates the corresponding masking coefficients

$$\{m_{i_c^{(j)}}^{(j)}\}_{\substack{1\le j\le n\\ 1\le c\le l}} := \mathsf{maskGenFast}\big((i_c^{(j)}, j),\ (p', q'),\ \mathsf{param}_{\mathcal{D}}\big)$$

Subsequently, it computes

$$\tilde{\tau}^{(j)} := \tau^{(j)} + \alpha \cdot \sum_{c=1}^{l} \nu_c\ m_{i_c^{(j)}}^{(j)}, \tag{5}$$

and then it verifies that the following equations hold

$$\tilde{\tau}^{(j)} \stackrel{?}{=} \begin{cases} \alpha\tilde{\mu}^{(j)} + \sum_{c=1}^{l} \nu_c\ \mathsf{PRF}(\mathsf{k_{prf}}, i_c^{(j)} \| j), \\ \qquad\qquad \text{if } 1 \le j \le k, \\ \alpha\tilde{\mu}^{(j)} + \sum_{c=1}^{l} \nu_c\ \mathbf{prf}_{i_c^{(j)}} \cdot \mathbf{G}^{(j)}, \\ \qquad\qquad \text{if } k+1 \le j \le n, \end{cases} \tag{6}$$

where $\mathbf{G}^{(j)}$ denotes the j^{th} column of generator matrix $\mathbf{G}$, and $\mathbf{prf}_{i_c^{(j)}} := \big(\mathsf{PRF}(\mathsf{k_{prf}}, i_c^{(j)} \| 1), \ldots, \mathsf{PRF}(\mathsf{k_{prf}}, i_c^{(j)} \| k)\big)$ is the vector of PRFs for segment $i_c^{(j)}$.

If the responses from all storage nodes $\{\mathcal{S}^{(j)}\}_{1\le j\le n}$ are well formed, algorithm U.Verify outputs $\mathsf{dec} := \mathsf{accept}$; otherwise it returns $\mathsf{dec} := \mathsf{reject}$.

$\mathsf{C.Repair}\ (^*\tilde{\mathcal{D}}, J_\mathsf{f}, \mathsf{param}_\mathsf{system}, \mathsf{param}_\mathcal{D}, \mathsf{maskGen}) \rightarrow (\tilde{\mathcal{D}})$:

On input of a corrupted data object $^*\tilde{\mathcal{D}}$ and a set of failed storage node indices $J_\mathsf{f} \subseteq [1,n]$, algorithm C.Repair first checks if $|J| > n-k+1$, i.e., the lost symbols cannot be reconstructed due to an insufficient number of remaining storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$. In this case, algorithm C.Repair terminates outputting $\perp$; otherwise, it picks a set of k surviving storage nodes $\{\mathcal{S}^{(j)}\}_{j\in J_\mathsf{r}}$, where $J_\mathsf{r} \subseteq [1,n] \setminus J_\mathsf{f}$ and, computes the masking coefficients $\{m_i^{(j)}\}_{\substack{j\in J_\mathsf{r}\\ 1\leq i\leq s}}$ and $\{m_i^{(j)}\}_{\substack{j\in J_\mathsf{f}\\ 1\leq i\leq s}}$, using the function maskGen, together with the parity check matrix $\mathbf{H} = [-\mathbf{P}^\top \mid \mathbf{I}_{n-k}]$.

Thereafter algorithm C.Repair unmasks the symbols held in $\{\mathcal{S}^{(j)}\}_{j\in J_\mathsf{r}}$ and reconstructs the original data object $\mathcal{D}$ using $\mathbf{H}$. Finally, algorithm C.Repair uses the generation matrix $\mathbf{G}$ and the coefficients $\{m_i^{(j)}\}_{\substack{j\in J_\mathsf{f}\\ 1\leq i\leq s}}$ to compute and subsequently mask the content of storage nodes $\{\mathcal{S}^{(j)}\}_{j\in J_\mathsf{f}}$.

Algorithm C.Repair then terminates by outputting the repaired data object $\tilde{\mathcal{D}}$.

5 SECURITY ANALYSIS

Req 0 : **Correctness.** We now show that the verification Equation 6 has to hold if algorithm C.Prove is executed correctly. In particular, Equation 6 consists of two parts: the first one defines the verification of the proofs $\{(\tilde{\mu}^{(j)}, \tau^{(j)})\}_{1\leq j\leq k}$ generated by the data storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq k}$; and the second part corresponds to the proofs $\{(\tilde{\mu}^{(j)}, \tau^{(j)})\}_{k+1\leq j\leq n}$ generated by the redundancy storage nodes $\{\mathcal{S}^{(j)}\}_{k+1\leq j\leq n}$. By definition the following equality holds:

$$\sigma_i^{(j)} = \alpha d_i^{(j)} + \mathsf{PRF}(\mathsf{k_{prf}}, i \parallel j),\ \forall 1 \leq i \leq s, 1 \leq j \leq k \quad (7)$$

We begin with the first part of Equation 6. By plugging Equations 5 and 4 to $\tilde{\tau}^{(j)}$ we get

$$\begin{aligned}\tilde{\tau}^{(j)} &= \tau^{(j)} + \alpha \cdot \sum_{c=1}^{l} \nu_c\, m_{i_c^{(j)}}^{(j)} \\ &= \sum_{c=1}^{l} \nu_c\, \sigma_{i_c^{(j)}}^{(j)} + \alpha \cdot \sum_{c=1}^{l} \nu_c\, m_{i_c^{(j)}}^{(j)}.\end{aligned}$$

Thereafter, by Equation 7 we get

$$\begin{aligned}\tilde{\tau}^{(j)} &= \sum_{c=1}^{l} \nu_c \left(\alpha d_{i_c^{(j)}}^{(j)} + \mathsf{PRF}(\mathsf{k_{prf}}, i_c^{(j)} \parallel j)\right) + \alpha \cdot \sum_{c=1}^{l} \nu_c\, m_{i_c^{(j)}}^{(j)} \\ &= \alpha \cdot \sum_{c=1}^{l} \nu_c \left(d_{i_c^{(j)}}^{(j)} + m_{i_c^{(j)}}^{(j)}\right) + \sum_{c=1}^{l} \nu_c\, \mathsf{PRF}(\mathsf{k_{prf}}, i_c^{(j)} \parallel j).\end{aligned}$$

Finally, by plugging Equations 1 and 3 we get

$$\tilde{\tau}^{(j)} = \alpha\tilde{\mu}^{(j)} + \sum_{c=1}^{l} \nu_c\, \mathsf{PRF}(\mathsf{k_{prf}}, i_c^{(j)} \parallel j).\ \square$$

As regards to the second part of Equation 6 that defines the verification of the proofs $\{(\tilde{\mu}^{(j)}, \tau^{(j)})\}_{k+1\leq j\leq n}$ generated by the redundancy storage nodes $\{\mathcal{S}^{(j)}\}_{k+1\leq j\leq n}$, we observe that for all $c \in [1,l]$ it holds that redundancy symbols $r_{i_c^{(j)}}^{(j)} = d_{i_c^{(j)}} \cdot \mathbf{G}^{(j)}$ and tags $\psi_{i_c^{(j)}}^{(j)} = \sigma_{i_c^{(j)}} \cdot \mathbf{G}^{(j)}$, whereby $\mathbf{G}^{(j)}$ is the j^{th} column of generator matrix $\mathbf{G}$, $d_{i_c^{(j)}} := (d_{i_c^{(j)}}^{(1)}, \ldots, d_{i_c^{(j)}}^{(k)})$ is the vector of data symbols for segment i_c, and $\sigma_{i_c^{(j)}} := (\sigma_{i_c^{(j)}}^{(1)}, \ldots, \sigma_{i_c^{(j)}}^{(k)})$ is the corresponding vector of linearly homomorphic tags. Hence, by Equation 7 the following equality always holds:

$$\begin{aligned}\psi_{i_c^{(j)}}^{(j)} &= (\alpha d_{i_c^{(j)}} + \mathbf{prf}_{i_c^{(j)}}) \cdot \mathbf{G}^{(j)} \\ &= \alpha r_{i_c^{(j)}}^{(j)} + \mathbf{prf}_{i_c^{(j)}} \cdot \mathbf{G}^{(j)},\end{aligned}$$

where $\mathbf{prf}_{i_c^{(j)}} := \left(\mathsf{PRF}(\mathsf{k_{prf}}, i_c^{(j)} \parallel 1), \ldots, \mathsf{PRF}(\mathsf{k_{prf}}, i_c^{(j)} \parallel k)\right)$ is the vector of PRFs for segment $i_c^{(j)}$. Thereby, given the same straightforward calculations as in the case of data storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq k}$, we derive the following equality:

$$\tilde{\tau}^{(j)} = \alpha\tilde{\mu}^{(j)} + \sum_{c=1}^{l} \nu_c\, \mathbf{prf}_{i_c^{(j)}} \cdot \mathbf{G}^{(j)}.\ \square$$

Req 1 : **Extractability.** We now show that POR-TOS ensures, with high probability, the recovery of an outsourced file D. To begin with, we observe that algorithms C.Prove and U.Verify can be seen as a distributed version of the algorithms SW.Prove and SW.Verify of the private PoR scheme in (Shacham, H. and Waters, B., 2008) executed across all storage nodes $\{\mathcal{S}^{(j)}\}_{1\leq j\leq n}$. More precisely, we assume that the MDS-code parameters (k,n) outputted by algorithm Setup , fulfill the retrievability requirements stated in (Shacham, H. and Waters, B., 2008), in addition to the reliability guarantee t.

We argue that given a sufficient number of interactions with a cheating cloud storage provider C', the user U eventually gathers linear combinations of at least $k \leq \rho \leq n$ code symbols for each segment of data object $\mathcal{D}$. These linear combinations are of the form

$$\tilde{\mu}^{(j)} \leftarrow \begin{cases}\sum_{c=1}^{l} \nu_c\, \tilde{d}_{i_c^{(j)}}^{(j)}, & \text{for } 1 \leq j \leq k \\ \sum_{c=1}^{l} \nu_c\, \tilde{r}_{i_c^{(j)}}^{(j)}, & \text{for } k+1 \leq j \leq n,\end{cases}$$

Table 2: Evaluation of the response time and the effort required by a storage node $\mathcal{S}$ to generate its response. The cheating cloud storage provider C' tries to deviate from the correct protocol execution in two ways: (i) by storing the data object $\mathcal{D}$ encoded but unmasked; and (ii) by not storing the data object $\tilde{\mathcal{D}}$ in its entirety. $\mathtt{RTT}$ is the round trip time between the user U and C; Π is the number of computations $\mathcal{S}$ can perform in parallel; and $\mathtt{T}_{\mathsf{puzzle}} := \mathcal{T} \cdot \mathtt{T}_{\mathsf{exp}}$ is the time required by $\mathcal{S}$ to generate one masking coefficient.

Scenario:	Response generation complexity for one $\mathcal{S}$	$\mathcal{S}$'s Response Time
Honest C:	$2l$ mult $+ 2(l+1)$ add	$\mathtt{T}_{\mathsf{resp}} = \mathtt{RTT} + \left\lceil \frac{2l}{\Pi} \right\rceil \mathtt{T}_{\mathsf{mult}} + \left\lceil \frac{2(l-1)}{\Pi} \right\rceil \mathtt{T}_{\mathsf{add}}$
C' stores $\mathcal{D}$ unmasked:	$l\mathcal{T}$ exp $+2l$ mult $+ (3l-2)$ add	$\mathtt{T}'_{\mathsf{resp}_1} = \mathtt{RTT} + \left\lceil \frac{l}{\Pi} \right\rceil \mathtt{T}_{\mathsf{puzzle}} + \left\lceil \frac{2l}{\Pi} \right\rceil \mathtt{T}_{\mathsf{mult}} + \left\lceil \frac{3l-2}{\Pi} \right\rceil \mathtt{T}_{\mathsf{add}}$
C' deletes up to $s(n-k+1)$ symbols from $\tilde{\mathcal{D}}$:	$\mathcal{S}$ with missing symbol: $\mathcal{T}$ exp $+2l$ mult $+ (2l+k-1)$ add; k $\mathcal{S}$s participating in symbol generation: $\mathcal{T}$ exp $+ (2l+1)$ mult $+ (2l-1)$ add; Remaining $\mathcal{S}$s: $2l$ mult $+ 2(l-1)$ add	$\mathtt{T}'_{\mathsf{resp}_2} = \mathtt{RTT} + \mathtt{T}_{\mathsf{puzzle}} + \left\lceil \frac{2l+1}{\Pi} \right\rceil \mathtt{T}_{\mathsf{mult}} + \left\lceil \frac{2l-1}{\Pi} \right\rceil \mathtt{T}_{\mathsf{add}}$

for known coefficients $(\nu_c)_{c=1}^{l}$ and known indices $i_c^{(j)}$ and j. Furthermore, U can efficiently derive the unmasked expressions

$$\mu^{(j)} \leftarrow \begin{cases} \sum_{c=1}^{l} \nu_c \, d^{(j)}_{i_c^{(j)}}, & \text{for } 1 \le j \le k \\ \sum_{c=1}^{l} \nu_c \, r^{(j)}_{i_c^{(j)}}, & \text{for } k+1 \le j \le n \end{cases}$$

by computing the masking coefficients $\{m^{(j)}_{i_c^{(j)}}\}_{\substack{1 \le j \le n \\ 1 \le c \le l}}$ using the function maskGenFast, and subtracting from $\tilde{\mu}^{(j)}$ the corresponding linear combination $\sum_{c=1}^{l} \nu_c \, m^{(j)}_{i_c^{(j)}}$.

Hereby, the extractability arguments given in (Shacham, H. and Waters, B., 2008) can be applied to the aggregated output of algorithms $\mathsf{C}.\mathsf{Prove}$ and $\mathsf{U}.\mathsf{Verify}$. More precisely, given that C' succeeds in making algorithm $\mathsf{U}.\mathsf{Verify}$ yield $\mathsf{dec} := \mathsf{accept}$ in an ε fraction of the interactions, and the indices $i_c^{(j)}$ of the challenge a chosen at random, then user U has at its disposal at least $\rho - \varepsilon > k$ correct code symbols for each segment of data object $\mathcal{D}$. Therefore, user U is able to reconstruct the data object $\mathcal{D}$ using the parity check matrix $\mathbf{H} = [-\mathbf{P}^{\top} \mid \mathbf{I}_{n-k}]$. □

Req 2 : **Soundness of Redundancy Generation.** In PORTOS, the cloud storage provider C is the party that generates the redundancy of the outsourced data object $\mathcal{D}$. Namely, the redundancy symbols and $\{r_i^{(j)}\}_{\substack{k+1 \le j \le n \\ 1 \le i \le s}}$ and their tags $\{\psi_i^{(j)}\}_{\substack{k+1 \le j \le n \\ 1 \le i \le s}}$ are computed by applying a linear combination over the original data symbols $\{d_i^{(j)}\}_{\substack{1 \le j \le n \\ 1 \le i \le s}}$ and their tags $\{\sigma_i^{(j)}\}_{\substack{1 \le j \le n \\ 1 \le i \le s}}$, respectively. Hence, following Theorem 4.1 in (Shacham, H. and Waters, B., 2008), if the pseudo-random function PRF is secure, then no cheating cloud storage provider C' will cause a verifier V to accept in a proof of data reliability instance, except by responding with values

$$\tilde{\mu}^{(j)} \leftarrow \sum_{c=1}^{l} \nu_c \, \tilde{r}^{(j)}_{i_c^{(j)}}, \quad \text{for } k+1 \le j \le n,$$

$$\tau^{(j)} \leftarrow \sum_{c=1}^{l} \nu_c \, \psi^{(j)}_{i_c^{(j)}}, \quad \text{for } k+1 \le j \le n.$$

that are computed correctly: i.e., by computing the pair $(\tilde{\mu}, \tau)$ using values $\tilde{r}^{(j)}_{i_c^{(j)}}$ and $\psi^{(j)}_{i_c^{(j)}}$ which are the output of algorithm $\mathsf{C}.\mathsf{GenR}$. □

Req 3 : **Storage Allocation Commitment.** We now show that a rational cheating cloud storage provider C' cannot produce a valid proof of data reliability as long as the time threshold $\mathtt{T}_{\mathsf{thr}}$ is tuned properly.

In essence, PORTOS consists of parallel proof of data possession challenges over all storage nodes $\{\mathcal{S}^{(j)}\}_{1 \le j \le n}$: a challenge for each symbol of the codeword. It follows that when a proof of data reliability challenge contains symbols which are not stored at rest, the relevant storage nodes cannot generate their part of the proof unless C' is able to generate the missing symbols. Hereafter, we analyze the effort that C' has to put in order to output a valid proof of data reliability in comparison to the effort an honest cloud storage provider C has to put in order to output the same proof. Given that the computational effort required by C and C' can be translated into their response time $\mathtt{T}_{\mathsf{resp}}$ and $\mathtt{T}'_{\mathsf{resp}}$, we can determine the lower and upper bounds for the time threshold $\mathtt{T}_{\mathsf{thr}}$.

A fundamental design feature of PORTOS is that C' has to compute a masking coefficient for each symbol of the encoded data object $\mathcal{D}$. We observe that the

Table 3: PORTOS computation, communication, and storage costs.

Scheme	PORTOS with Symbol Tags
U.Store complexity:	sk PRF $+ sk$ mult $+ sk$ add
C.Prove complexity:	n $\mathsf{PRG}_{\mathsf{chal}} + 2nl$ mult $+ 2n(l+1)$ add
U.Verify complexity:	n $\mathsf{PRG}_{\mathsf{chal}} + 2nl$ exp $+ kl(n-k+1)$ PRF $+2n(l+1) + kl(n-k)$ mult $+ (n-k)(kl+k+2)$ add
Storage cost:	$2\times$ the size of $\mathcal{D}$
Bandwidth:	$2n$ symbols

masking coefficients have the same size as $\mathcal{D}$'s symbols. Hence, assuming that $\mathcal{D}$ cannot be compressed more (e.g because it has been encrypted by the user), a strategy whereby C' is storing the masking coefficients would effectively double the required storage space. Moreover, a strategy whereby C' does not store the content of up to $n-k+1$ storage nodes and yet it stores the corresponding masking coefficients, would increase C''s operational cost without yielding any storage savings. Given that strategies that rely on storing the masking coefficients do not yield any gains in terms of either storage savings or overall operational cost, C' is left with two reasonable ways to deviate from the correct protocol execution:

(i) The first one is to store the data object $\mathcal{D}$ encoded but unmasked. Although this approach does not offer any storage savings, it significantly reduces the complexity of storing and maintaining $\mathcal{D}$ at the cost of a more expensive proof generation. More specifically, in order to compute a PORTOS proof, C' has to generate $2l$ masking coefficients $\{m_{i_c^{(j)}}^{(j)}\}_{\substack{1\le j\le n\\ 1\le c\le l}}$.

(ii) The second way C' may misbehave, is by not storing the data object $\tilde{\mathcal{D}}$ in its entirety and hence generating the missing symbols involved in a PORTOS challenge on-the-fly. In particular, C' can drop up to $s(n-k+1)$ symbols of $\tilde{\mathcal{D}}$ either by not provisioning up to $n-k+1$ storage nodes; or by uniformly dropping symbols from all n storage nodes $\{\mathcal{S}^{(j)}\}_{1\le j\le n}$, ensuring that it preserves at least k symbols for each data segment.

In order to determine the lower bound for the time threshold $\mathsf{T}_{\mathsf{thr}}$ we evaluate the response time $\mathsf{T}_{\mathsf{resp}}$ of an honest cloud storage provider C. Additionally, for each type of C''s malicious behavior we evaluate its response time, and determine the upper bound for the time threshold $\mathsf{T}_{\mathsf{thr}}$ as $\mathsf{T}'_{\mathsf{resp}} = \mathsf{min}(\mathsf{T}'_{\mathsf{resp}_1}, \mathsf{T}'_{\mathsf{resp}_2})$, where $\mathsf{T}'_{\mathsf{resp}_1}$ and $\mathsf{T}'_{\mathsf{resp}_2}$ denote C''s response time when it opts to keep $\mathcal{D}$ unmasked and delete symbols of $\tilde{\mathcal{D}}$, respectively. Concerning the evaluation of $\mathsf{T}'_{\mathsf{resp}_2}$, we consider the most favorable scenario for C' where it has to generate only one missing symbol for a PORTOS challenge. Table 2 presents the effort required by a storage node $\mathcal{S}$ in order to output its response, for each of the scenarios described above, together with the corresponding response time. For the purposes of our analysis, we assume that all storage nodes $\{\mathcal{S}^{(j)}\}_{1\le j\le n}$ have a bounded capacity of Π concurrent threads of execution, that computations —exponentiations, multiplications, additions, etc.— require a minimum execution time, and that $\mathsf{T}_{\mathsf{add}} \ll \mathsf{T}_{\mathsf{exp}}$ and $\mathsf{T}_{\mathsf{add}} \ll \mathsf{T}_{\mathsf{mult}}$. Furthermore, we assume that $\{\mathcal{S}^{(j)}\}_{1\le j\le n}$ are connected with premium network connections (low latency and high bandwidth), and hence the communication among them has negligible impact on C' response time. As given in Table 2, *Req 3* is met as long as the time threshold $\mathsf{T}_{\mathsf{thr}}$ is tuned such that it fulfills the following relations:

$$\mathsf{T}_{\mathsf{thr}} > \mathtt{RTT}_{\mathsf{max}} + \left\lceil \frac{2l}{\Pi} \right\rceil \mathsf{T}_{\mathsf{mult}} \text{ (Lower bound)},$$

$$\mathsf{T}_{\mathsf{thr}} < \mathtt{RTT}_{\mathsf{max}} + \mathsf{T}_{\mathsf{puzzle}} + \left\lceil \frac{2l+1}{\Pi} \right\rceil \mathsf{T}_{\mathsf{mult}} \text{ (Upper bound)},$$

where $\mathtt{RTT}_{\mathsf{max}}$ is the worst-case RTT and $\mathsf{T}_{\mathsf{puzzle}} := \mathcal{T} \cdot \mathsf{T}_{\mathsf{exp}}$ is the time required by $\mathcal{S}$ to evaluate the function maskGen. By carefully setting the puzzle difficulty coefficient $\mathcal{T}$, we can guarantee that $\mathsf{T}_{\mathsf{resp}} \ll \mathsf{T}'_{\mathsf{resp}}$ and $\mathtt{RTT}_{\mathsf{max}} \ll \mathsf{T}'_{\mathsf{resp}} - \mathsf{T}_{\mathsf{resp}}$, and hence make our proof of data reliability scheme robust against network jitter. Finally, notice that PORTOS can adapt to C''s computational capacity as it evolves over time by tuning $\mathcal{T}$ accordingly. □

6 PERFORMANCE ANALYSIS

Table 3 summarizes the performance of PORTOS in terms of computation, communication, and storage costs. The computational effort required to verify the response of a redundancy node $\{\mathcal{S}^{(j)}\}_{k+1\le j\le n}$ is much higher than the effort required to verify the response of a data node $\{\mathcal{S}^{(j)}\}_{1\le j\le k}$. Namely, in addition to the $2nl$ exponentiations required by the function maskGenFast to generate the masking coefficients of the nl involved symbols, algorithm U.Verify evaluates kl PRFs and $k+2(l+1)$ multiplications in

order to verify the response of each redundancy node, compared to the l PRFs and $2(l+1)$ multiplications it has to compute for the response of each data node. We conclude that compared to existing erasure-code based proof of data reliability schemes, namely (Bowers et al., 2009; Chen et al., 2015; Vasilopoulos et al., 2018), we achieve comparable computational gain while enabling data repair at the cloud side. Nevertheless, we observe that storage and bandwidth costs remain important. In order to improve the performance of our scheme and reduce these costs, we propose a new version of PORTOS which namely implements the storage efficient variant of the linearly homomorphic tags introduced in (Shacham, H. and Waters, B., 2008). More specifically, instead of generating one tag per symbol, the new algorithm U.Store computes a linearly homomorphic tag for a data segment, comprising k symbols. Due to space constrains, we omit the description of this new solution which will be included in an extended technical report, that will become available after the review process.

7 CONCLUSION

In this paper, we proposed PORTOS, a novel proof of data reliability solution for erasure-code-based distributed cloud storage systems. PORTOS enables users to verify the retrievability of their data, as well as the integrity of its respective redundancy. Moreover, in PORTOS the cloud storage provider generates the required redundancy and performs data repair operations without any interaction with the user, thus conforming to the current cloud model. Thanks to the combination of PDP with time-lock puzzles, PORTOS provides a *rational* cloud storage provider with a strong incentive to provision sufficient redundancy, which is stored *at rest*, guaranteeing this way a reliable storage service.

REFERENCES

Armknecht, F., Barman, L., Bohli, J.-M., and Karame, G. O. (2016). Mirror: Enabling proofs of data replication and retrievability in the cloud. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16.

Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., and Song, D. (2007). Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07.

Barsoum, A. F. and Hasan, M. A. (2012). Integrity verification of multiple data copies over untrusted cloud servers. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '12.

Barsoum, A. F. and Hasan, M. A. (2015). Provable multicopy dynamic data possession in cloud computing systems. *IEEE Transactions on Information Forensics and Security*, 10.

Blaum, M., Brady, J., Bruck, J., and Menon, J. (1994). Evenodd: An optimal scheme for tolerating double disk failures in raid architectures. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, ISCA '94.

Bowers, K. D., Juels, A., and Oprea, A. (2009). Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09.

Bowers, K. D., van Dijk, M., Juels, A., Oprea, A., and Rivest, R. L. (2011). How to tell if your cloud files are vulnerable to drive crashes. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11.

Chen, B., Ammula, A. K., and Curtmola, R. (2015). Towards server-side repair for erasure coding-based distributed storage systems. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15.

Chen, B. and Curtmola, R. (2013). Towards self-repairing replication-based storage systems using untrusted clouds. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13.

Chen, B. and Curtmola, R. (2017). Remote data integrity checking with server-side repair. *Journal of Computer Security*, 25.

Chen, B., Curtmola, R., Ateniese, G., and Burns, R. (2010). Remote data checking for network coding-based distributed storage systems. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10.

Curtmola, R., Khan, O., Burns, R., and Ateniese, G. (2008). Mr-pdp: Multiple-replica provable data possession. In *Proceedings of the 28th International Conference on Distributed Computing Systems*, ICDCS '08.

Erway, C., Küpçü, A., Papamanthou, C., and Tamassia, R. (2009). Dynamic provable data possession. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09.

Etemad, M. and Küpçü, A. (2013). Transparent, distributed, and replicated dynamic provable data possession. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*, ACNS'13.

Juels, A. and Kaliski, Jr., B. S. (2007). Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07.

Le, A. and Markopoulou, A. (2012). Nc-audit: Auditing for network coding storage. In *Proceedings of International Symposium on Network Coding*, NetCod '12.

Leontiadis, I. and Curtmola, R. (2018). Secure storage with replication and transparent deduplication. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18.

Rivest, R. L., Shamir, A., and Wagner, D. A. (1996). Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA.

Shacham, H. and Waters, B. (2008). Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08.

Suh, C. and Ramchandran, K. (2011). Exact-repair mds code construction using interference alignment. *IEEE Trans. Inf. Theor.*, 57(3).

Thao, T. P. and Omote, K. (2016). Elar: Extremely lightweight auditing and repairing for cloud security. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16.

Vasilopoulos, D., Elkhiyaoui, K., Molva, R., and Onen, M. (2018). Poros: Proof of data reliability for outsourced storage. In *Proceedings of the 6th International Workshop on Security in Cloud Computing*, SCC '18.

Xing, C. and Ling, S. (2003). *Coding Theory: A First Course*. Cambridge University Press, New York, NY, USA.