Implementation and Evaluation of Publicly Verifiable Proofs of Data Replication and Retrievability for Cloud Storage

---

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Master of Science
in the
University of Canterbury
by
Hao Li

University of Canterbury
2022

Abstract

Cloud storage is a mature and widely used cloud technology. Behind this, it involves the proofs of the retrievability of files stored on the server, which allows the client to remotely check whether its files are correctly stored on the cloud server. In 2020, Gritti proposed the first publicly verifiable Proofs of Retrievability and Reliability (P-PORR). P-PORR combined Proofs of Retrievability (POR) with Verifiable Delay Functions (VDF) providing a fast verification and allowing anyone to verify the server's behavior, not just the client. We built a realistic cloud test environment, implemented and tested P-PORR in this environment. This paper describes the implementation process, analyzes the performance of P-PORR from multiple perspectives, such as the total time spent before files are uploaded to the server, verification time and financial considerations. We stand in the perspective of the client and server to discuss the effect of the results. The results show that P-PORR has a similar performance to another PORR, with private verification, called Mirror. It even outperforms Mirror in processing small files. Therefore, P-PORR is suitable for cloud storage services. In the long run, this paper provides a practical application instance of VDF and a performance comparison benchmark for later PORR research.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I received a lot of assistance and support during the writing of this thesis.

I would first like to thank my senior supervisor, Dr Clementine Gritti, for her passion and expertise in my field of study, for her work in formulating the research question, valuable advice on methodology and for writing this thesis. Her insightful and deep feedback inspired me and took my work to a higher level.

I would like to thank my co-supervisor, Professor Andreas Willig, for giving me all further research opportunities, and the right guidance when setting up my cloud testing environment, allowing me to do my best work with limited time.

I would like to thank Kat Bell and Sophia Rabara for showing enough concern for my life and helping me with any computer hardware issues.

Finally, I would like to thank my parents for their love, determination and support. As an international student, it would have been completely impossible for me to complete this dissertation without their selfless dedication.

Chapter I

Introduction

1.1 Background

Internet-based cloud services are widely welcomed by individuals and industries, because of their obvious advantages such as convenience, security, and low cost. It can reduce capital expenditures and convert them into operating costs [1]. As a common and important cloud service, cloud storage is also accepted by many parties. The cloud storage server can flexibly store and access files according to the particular type of service requested. Because the cloud provider has absolute control over the server, the behavior of the server and the cloud provider can be understood as consistent, and the cloud provider or the server mentioned later is the same party. Due to the value of data itself, cloud storage servers have become vulnerable targets. However, the risk from cloud storage often includes more than external hacker attacks [2], misoperation and misconfiguration of internal users can also lead to catastrophic consequences. For example, a company called Front Edge CNC found that the online production data which had been saved in Tencent cloud storage was completely lost, the estimated incident resulted in the loss of nearly 10 million RenMinBi(RMB) in the platform data. For this reason, Tencent cloud stated that the incident was caused by a Silent Data Corruption (inconsistent writing and reading data) error caused by a bug in the firmware version of the physical hard disk, and the metadata of the file system was corrupted [3, 4].

Therefore, disaster recovery is a basic function to assure customers. As a backup method, cloud file replicas make storing cloud files more secure, these replicas are usually stored in different places. When cloud files are damaged or attacked by ransomware, replicas can effectively prevent file loss. For instance,

since the original file and the replicas are stored on different servers, even if some servers fail, data recovery and availability can be ensured. Generally speaking, cloud service providers charge users based on the redundancy level (file recovery capability) [5]. For instance, storage services such as Amazon S3 and Google FS can withstand up to two concurrent server failures [6].

1.2 Problem Statement

The key characteristics of files stored in the cloud are retrievability and integrity. Even large cloud companies have vicious incidents related to these characteristics [2, 7], such as the Tencent Cloud mentioned before. Such incidents mean that retrievability and integrity cannot be easily guaranteed, which is the main reason why users do not trust cloud storage. Previous research has defined two schemes: Proof of Data Possession (PDP) [8, 9] and Proof of Retrievability (POR) [10, 11]. They both provide integrity assurance for clients' files stored in the cloud. PDP enables the client to verify that an untrusted server is storing the client's data without accessing the entire file from the server. Cryptographic integrity assurance allows clients to detect unauthorized modifications to partial files upon retrieval. This basic form of integrity assurance cannot detect modification or deletion of files before they are retrieved or on an ongoing basis. POR aims to provide this higher level of assurance [10].

The variants of the PDP schemes also check for replicas of files [12, 13, 14]. However, existing solutions require the client to create replicas of the file itself, perform appropriate preprocessing on the replicas, and finally upload all processed replicas with the file to the server. In uploading original files and replicas to the server, cloud providers consume much bandwidth to process requests as the number of replicas increases. For cloud providers, much bandwidth means increased operating costs [15].

In the actual operation process, there will be some unavoidable malicious behaviors. The client seems to have implicit trust in the cloud provider and cloud technology, and there is negligence in reading the service level agreements [16, 17]. The cloud provider can take advantage of the client's human behavior to act dishonestly. For example, the files or replicas were not

stored according to the agreements. Some malicious clients may abuse their rights by generating replicas locally and uploading fake replicas to gain more storage space. For example, Amazon S3 charges its clients nearly 25% of the basic storage cost required for additional replication [17, 18], so it is cheaper to upload fake replicas than other original files. Since the data uploaded by the clients is usually encrypted, the provider cannot identify whether the uploaded content is indeed a replica.

1.3 Motivation

Armknecht et al. addressed the issues mentioned above in [15] and extended POR, called Proof of Retrievability and Reliability (PORR). PORR supports correct storage verification of files and replicas and allows the server to build replicas, which will effectively avoid similar behaviors of malicious clients seeking economic benefits by uploading fake replicas, and can reduce the bandwidth cost of the server. However, the PORR system in [15], called Mirror, supports private verification, only the client has permission to check that the server has correctly stored its files and their replicas. Gritti proposed a new public PORR protocol in [16], we call it P-PORR. P-PORR gives the same storage correctness guarantee as PORR and provides public verification for the client. Anyone (person/organization) can act as a verifier to check whether the server stores the client's data correctly. P-PORR has been theoretically proven feasible, but has not been implemented and applied in a cloud environment to test its performance.

A theoretically feasible protocol does not bring practical benefits to humans. We first need to consider whether the functions of this protocol are feasible in a realistic cloud environment, followed by user experience, security and financial considerations. In continuous implementation and testing, we will gradually discover the strengths and shortcomings of the protocol. We can actively use its strengths to appropriate areas and improve the shortcomings exposed in the test. As the first PORR protocol that supports public verification, we focus on whether P-PORR satisfies the client's security needs and the server's financial considerations. The test results will bring great reference metrics to future researchers in related fields.

1.4 Research Question

This research aims to implement and evaluate P-PORR by designing a realistic cloud framework and executing multiple benchmarks of the solution, comparing the experimental results obtained with the existing private PORR in the cloud, namely Mirror. Therefore, the research question of this study is:

Does the evaluation of the new prototype proposed in [16] for proving data replication and retrievability in the cloud show acceptable performance results in a realistic cloud setting?

1.5 Publication

The work of this Master's research has been accepted for publication by the Advances in Science, Technology and Engineering Systems Journal (ASTESJ).

1.6 Thesis Structure

The structure of the thesis is presented as follows:

- **Chapter 2** provides a literature survey related to this paper, and background knowledge is introduced to help readers better understand the contents of this paper.
- **Chapter 3** describes each algorithm of the protocol P-PORR.
- **Chapter 4** describes how we set up a realistic cloud test environment and how we implement P-PORR.
- **Chapter 5** analyzes our experimental results and compares them with existing work.
- **Chapter 6** concludes the research implications, limitations, and future work of this paper.

Chapter II

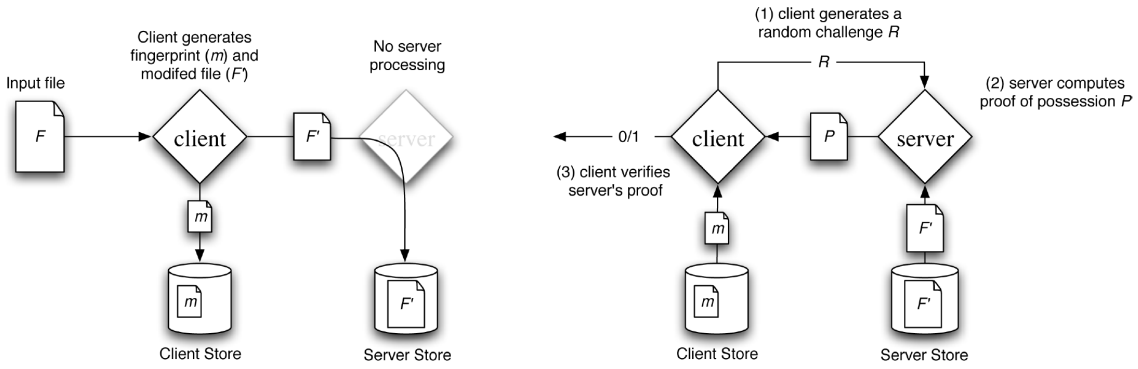
Literature Review

This chapter introduces the research progress of related work in the same field, as well as various variants of the same scheme. Different protocols usually offer particular features, such as protecting client data integrity, retrievability, seamless access to dynamic data, and public verification.

2.1 Proof of Data Possession

Ateniese et al. introduced the concept of Proof of Data Possession (PDP) [8, 9]. PDP is concerned with verifying whether an untrusted server stores client data without retrieving data from the server. There is also no need for the server to access the entire file. As shown in Figure 2.1, the client (data owner) preprocesses the file, which includes generating metadata (fingerprint) stored locally, and Homomorphic Verifiable Tags (HVTs) are added to each data block of the file. The processed file is finally uploaded to the server. During the verification process, the client first generates a challenge, in which the client specifies the blocks for which it wants possession proof, then sends the challenge to the server. The server uses the public key to generate a possession proof for the block specified in this challenge and returns it to the client. The client will finally verify the proof with the metadata (fingerprint) and public and private keys kept locally to determine whether the server has kept the file.

The above method belongs to the classic PDP concept, which uses asymmetric cryptography. The public key and private key are required in the protocol, which improves the security, but when PDP is used to prove that it has a large amount of data, it will bring a lot of large I/O and computational burden to the server [8, 9]. In summary, [8, 9] lay a theoretical foundation for later related work, but there is still space for improvement.



(a) Pre-process and store (b) Verify file possession by server

Figure 2.1: Protocol for provable data possession

Ateniese et al. constructed an efficient and provably secure PDP technique based entirely on symmetric key cryptography [19] based on the problems exposed in previous studies. Compared with previous PDP techniques, this version of the PDP technique allows uploading of dynamic data, it effectively supports operations such as block modification, deletion, and appending. Supporting such operations means that this version of PDP is more practical, as many cloud storage services are not limited to static or warehouse data. For example, multiple authorized users (users who have the right to access the client's file) in Figure 2.2 want to access the files stored in the cloud server by the client at the same time, dynamic data operations are required in this scenario. There is another difference from the previous works is that the scheme in [3] is completely based on symmetric key cryptography, and the symmetric encryption algorithm operates fast, it requires less computational resources, but is not suitable for public (third-party) verification.

The main idea of [19] is that the client precomputes a certain number of short possession verification tokens before uploading, each covering some set of data blocks. When the client wants to obtain proof of data possession, it selects a random set of block indexes and challenges the server, which must compute a short integrity check on the specified block (corresponding to the index) and assign it to return to the client. The returned integrity check must match the corresponding value precomputed by the client for the proof

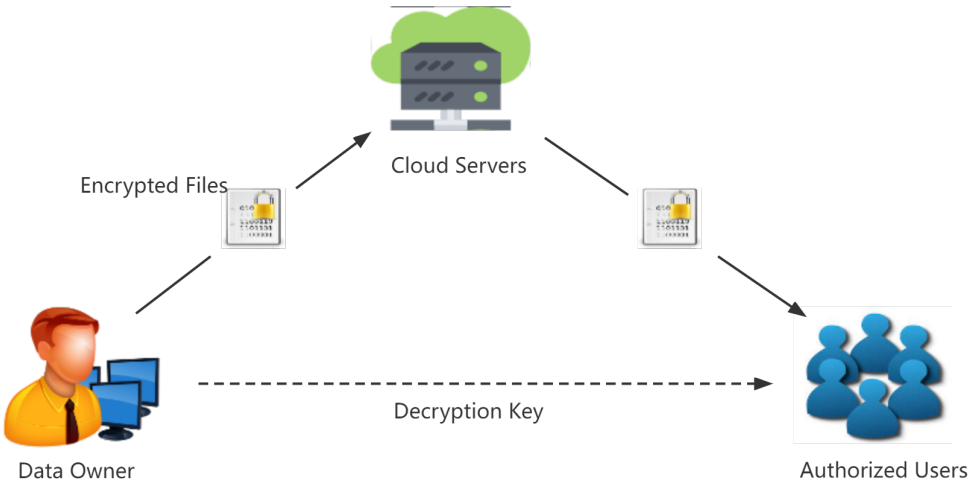


Figure 2.2: Dynamic cloud data storage model

to hold. However, the disadvantage of this scheme is that a fixed number of verification will be set when the protocol is initialized, that is to say, the client cannot send unlimited challenges to the server. If we want to increase the number of verification, we will need to re-initialize the protocol. In turn, the data needs to be re-uploaded to the server. Nevertheless, the authors argues that is not a problem in practice and with the default settings of this protocol, it enables the client to verify possession every 15 minutes for the next 16 years [19].

In recent years, due to the increasing demand for multiple replicas of cloud storage, the proof of data possession is not only limited to the file itself also extended to the replicas of the file [12, 13, 14]. Curtmola et al. first proposed a Multi-Replicas Provable of Data Possession (MR-PDP) scheme that extends previous work on PDP for a single file in a client/server storage system [8]. MR-PDP can create multiple replicas of client's files and verify them. The MR-PDP scheme improves data availability, and corrupted replicas of data can be rebuilt using replicas on other servers at an acceptable cost. However, the MR-PDP scheme only supports private verification, that is, only the client can check the possession of the data and does not consider the interaction between authorized users and cloud service providers. Public verifiability is a

crucial feature of remote data inspection schemes. Possible disputes between client and cloud service providers can be avoided. Delegating the verification process to a trusted and professional third party for data integrity will reduce the likelihood of exposing keys, also resolve such disputes.

Ayad et al. proposed a Pairing-Based Provable Multi-Replica Data Possession (PB-PMDP) scheme that provides clients with proof that all replicas are stored and kept intact [13]. In addition, it allows authorized users to access replicas of files stored by cloud service providers seamlessly, supports public verifiability, and allows unlimited verification. The structure can be referred to Figure 2.2.

The overall process of this scheme is similar to the standard PDP [8, 9], the main difference is that the core of this scheme is to generate unique differentiable replicas for data files. Identical data replicas enable cloud service providers to trick the client into storing only one replica and pretend it stores multiple replicas. PB-PMDP utilizes the diffusive properties of any secure encryption scheme to generate different replicas to deal with similar misbehaviour. If there is a single bit change in the plaintext, there will be an unpredictable complete change in the ciphertext. At the same time, PB-PMDP also introduces the concept of Homomorphic Linear Authenticators (HLAs) in Proof of Retrievability (POR) [11, 20, 21], the research content of POR will be introduced in detail later. The experimental results of PB-PMDP show that the performance of PB-PMDP is better than that of MR-PDP, which is reflected in the computing time of cloud service providers and the verification time of verifiers. Especially in the latter case, the time in the PB-PMDP scheme does not change much due to the increase in the number of replicas. The PB-PMDP scheme has high computational efficiency in verifying a large number of file replicas.

In a later study, Ayad et al. proposed a Mapping-Based Provable Multi-Replica Dynamic Data Possession (MB-PMDDP) scheme [14]. It continued the features of the previous PB-PMDP [13], authorizes users to access replicas of files stored by cloud service providers, exposes public verifiability, and allows unlimited verification. The difference is that MB-PMDDP enables the client to update and extend blocks of file replicas uploaded to servers. Verifying such replicas of dynamic data requires the knowledge of the block

versions to ensure that the data blocks in all replicas are identical with the most recent modifications issued by the client. The verifiers should be familiar with the block indices to guarantee that the server inserted or added a new block at the requested position in all replicas. To this end, this scheme is based on using a small data structure (metadata), which is called a map-version table. This scheme is the first to handle multiple replicas of dynamic data.

2.2 Proofs of Retrievability

Proofs of Retrievability (POR) is similar to PDP but stronger, because it supports the data recovery [12]. It was first proposed by Juels and Kaliski [10]. POR is mainly concerned with data retrievability and data integrity, which enables the server to generate concise proof to prove to the client can retrieve a specific file, that is, the server retains and transmits the file data reliably enough to allow the user to recover the entire file. The existing cryptographic integrity guarantees allow verifiers to detect unauthorized modifications to parts of a file when retrieving files, such as various variants of PDPs. However, this basic form of integrity assurance cannot detect modification or deletion of files prior to file retrieval or on an uninterrupted basis. The goal of POR is to provide this higher level of assurance on remote files without requiring users to download the file themselves [10].

The original POR protocol [10] will generate a key in the beginning, and this key will be kept locally by the verifier for later challenge-response protocol. This key is also used to encrypt the file and randomly embeds a set of check blocks of random values called sentinels. The use of encryption here makes it impossible for the server to distinguish the sentinel from other data blocks. The verifier challenges the server by specifying the sentinel set's location and asking the server to return the relevant sentinel value. If the server modifies or deletes part of the file, it is most likely that sentinels are also included, therefore unlikely to respond to the verifier correctly. The model is shown in Figure 2.3. POR also employs error-correcting codes to avoid a particular part of the file being corrupted by the server due to unknown factors, significantly increasing the protocol's robustness. One disadvantage

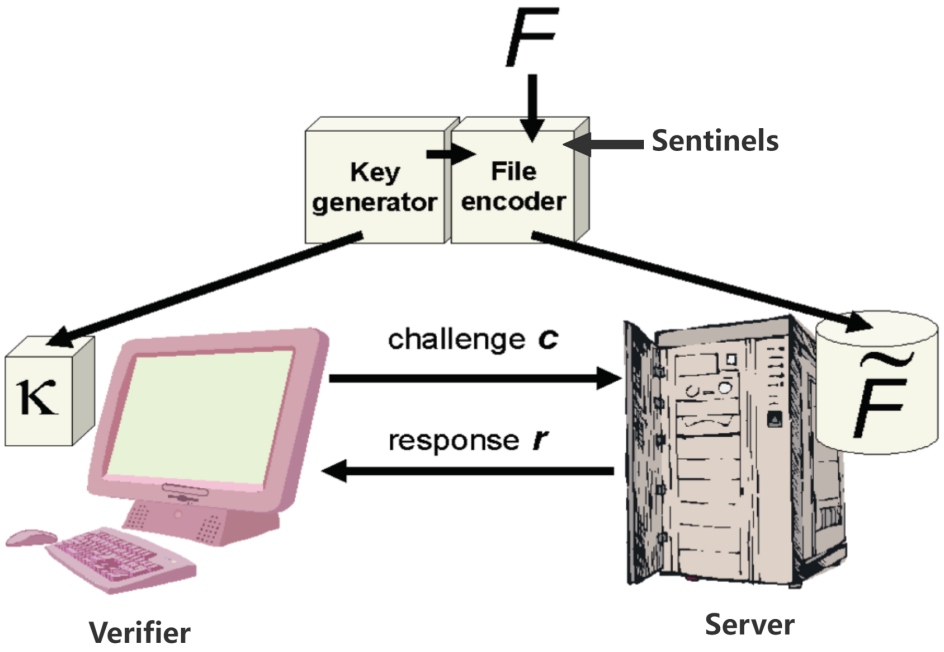


Figure 2.3: POR sentinel system model

of the POR protocol is that the server needs to preprocess/encode the file before uploading it to the cloud (besides the overhead of simple encryption or hashing), which adds some extra computational overhead. Embedding of sentinels will occupy about 2% of the space of the encoded file, as well as error-correcting codes, which increases a lot of storage overhead.

The recent research on POR gradually made up for some of the weaknesses of sentinels-based POR. For example, sentinels-based POR is mainly used for static data storage and does not support public verification. Shacham and Waters gave two POR schemes [11, 22]. The first scheme is built from Boneh–Lynn–Shacham (BLS) signature and secure in the random oracle model, it has the shortest query and response of any POR scheme with public verifiability. The second scheme is built on a pseudorandom Function (PRF). It is secure in the standard model, with the shortest responses (but longer queries) of any POR scheme with private verifiability. Both schemes rely on homomorphic properties to aggregate proofs into a small authenticator value. To help with better understanding, here is an introduction to the standard model and the random oracle model:

- Standard model is a computational model where the adversary is limited by the available time and computational power. Since cryptographic schemes are usually based on complexity assumptions, some problems, such as factorization, cannot be solved in polynomial time. A scheme that is provably secure using only complexity assumptions is called a secure in the standard model [23].

- Random oracle model is a black box in theory that responds to each unique query with a truly random response uniformly chosen from its output domain. If the query is repeated, it will respond the same way to the submitted query every time. In other words, a random oracle is a mathematical function that is uniformly chosen at random, that is, a function that maps every possible query to a fixed random response from its output domain [24].

The two POR schemes of Shacham and Waters expand the adaptability of the POR scheme when dealing with different needs. They provides the shortest query and response time but still takes up a lot of storage space. Because an equal length authenticator accompanies each data block, this adds two times the overhead over erasure codes. The server's response in the POR protocol is two times the length of the authenticator [11, 22]. The increase of storage undoubtedly increases the operating cost of the server, which is unfavourable for the server.

The recent studies not only focus on the retrievability of static data only, they also focuses on dynamic data [25, 26]. Qian et al. performed block tag authentication by manipulating the classical Merkle Hash Tree (MHT) construction [25]. It improves the original POR model [10] and supports public verifiability and dynamic data manipulation. Specifically, the original POR scheme does not consider dynamic data manipulation and does not support block insertion at all.

This is because the construction of the signature involves file index information. Therefore, the computational overhead is unacceptable once a file block is inserted since the signatures of all the following file blocks should be recomputed with the new index. In order to solve this limitation, Qian et al. delete the index information when generating the signature [25], use

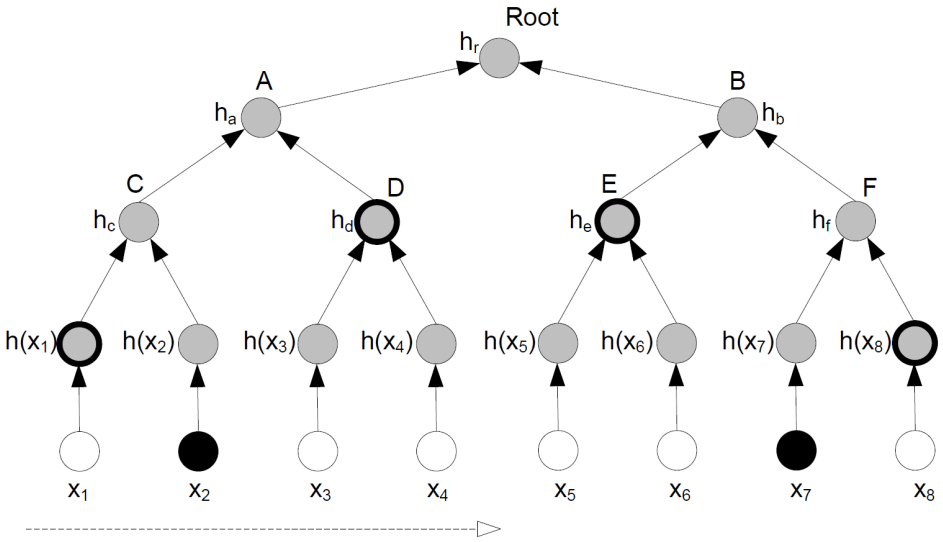


Figure 2.4: Merkle hash tree authentication of data elements

the hash value of the data block as the label of this block, then store it on the leaf of the MHT, finally use the MHT to verify the value and location of the data blocks. So individual data operations on any data block do not affect others. MHT structure can refer to Figure 2.4 that depicts an example of verification. We assume the two black filled circles $\{x_2, x_7\}$ are the received blocks for verification, the verifier firstly computes the $h(x_2)$, $h(x_7)$, $h_c = h(h(x_1)||h(x_2))$, $h_f = h(h(x_7)||h(x_8))$, $h_a = h(h_c||h_d)$, $h_b = h(h_e||h_f)$, and $h_r = h(h_a||h_b)$, then checks whether if the calculated h_r/root is the same as the expected one.

2.3 Verifiable Delay Functions

Verifiable Delay Functions (VDF) were first introduced by Boneh et al. [27]. VDF requires a specified number of sequential steps to evaluate, and any parties with parallel computing resources or specialized hardware cannot shorten the evaluation time. Finally, a unique output is generated that can be publicly and efficiently verified. The model is shown in Figure 2.5. The three specific algorithms of VDF are as follows:

- $Setup(\lambda, t) \rightarrow$ public parameters $pp =$ (evaluation key ek, verification key vk)

is a randomized algorithm. The required input parameters are a security parameter λ and a desired puzzle difficulty t. The produced output is public parameters pp, which consists of an evaluation key ek and a verification key vk. VDF requires *Setup(λ, t)* to be polynomial-time within λ. By convention, public parameters specify an input space X and an output space Y. *Setup(λ, t)* may require secret randomness, resulting in a scheme requiring a trusted setup. In order to achieve meaningful safety, the puzzle difficulty t is constrained to be sub-exponentially sized in λ.

- $Eval(ek, x) \rightarrow \{\text{output y}, \text{proof } \pi\}$ is a deterministic algorithm. It takes an input (puzzle) $x \in X$, produces an output $y \in Y$ and a proof π. *Eval(ek, x)* can generate the proofs π using random bits, but not to compute y. For all *pp* generated by *Setup(λ, t)* and all $x \in X$, the algorithm *Eval(ek, x)* must run in parallel time t of the $poly(log(t), \lambda)$ processors.

- $Verify(vk, x, y, \pi) \rightarrow \{Yes, No\}$ is a deterministic algorithm. The required input parameters are the verification key vk, the puzzle x and its solution y, and the proof π. If the y is an expected output of x, the output of *Verify(vk, x, y, π)* would be "Yes"; otherwise "No". *Verify(vk, x, y, π)* must run in the total time polynomial of $logt$ and λ. We notice that *Verify(vk, x, y, π)* is much faster than *Eval(ek, x)*.

VDF can be applied in a wide range of scenarios, such as decentralized applications, Internet of Things (IoT), network attack detection and block chain [28, 29, 30]. As early as the 1990s, Rivest et al. designed a time-lock puzzle [31], which involves computing an inherently sequential function whose purpose is to encrypt a message so that no one, not even the sender, can decrypt it until a predetermined time has elapsed, even if a continuously running computer at least cannot be solved such computational problems for a certain time either. The only solution is to use repeated squaring in the RSA group as a time-locked puzzle. However, time-lock puzzles are not required to be universally verifiable, and the verifier uses its secret state to prepare each puzzle and verify the result. In contrast, a VDF may require

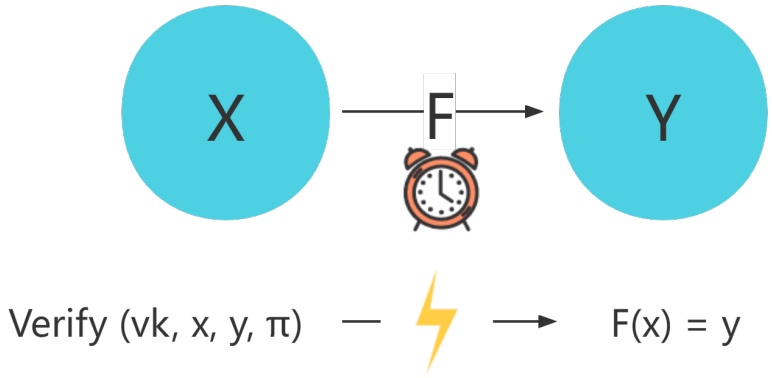


Figure 2.5: Verifiable Delay Functions model

an initial trusted setup, but then must be able to be used for any randomly chosen input [27].

Attias et al. proposed a rate-limiting protocol based on a modular exponential VDF [29], which can reduce network overhead and provide protection for transaction generation between nodes in IoT networks. Nodes in IoT networks can be an IoT device or a smartphone, each node participates in the generation and verification of transactions, which transfer tokens between two nodes, or just carry data. Since this kind of transaction generation is not limited by any fees, if there is no proper access control mechanism, a node may theoretically generate an infinite number of transactions per second, that is, this node is being DoS attacked, resulting in network interruption. Experimental results by Attias et al. demonstrate that VDF can also be used to prevent denial of service attacks in IoT networks [29], which is an instance of applying a mathematical framework to real-world problems.

VDF can also be applied to the block chain field. Zhou et al. found that the Difficulty Adjustment Algorithm (DAA) embedded in the proof of work (PoW) block chain algorithm often fluctuates when generating blocks [30], resulting in the generated blocks either not generating blocks for a long time, or generating multiple blocks in a short time. In addition, this strategy will also lead to potential attacks on DAA (such as selfish attacks, jumping mining attacks, etc.) [30]. To address these issues, Zhou et al. propose an

improved block chain consensus protocol that utilizes PoW with Distributed Verifiable Delay Function (DVDF). This protocol can retain the proof-of-work mechanism to ensure the probability of finding a suitable block. Furthermore, using DVDF, this protocol can ensure that all miners perform a computation that takes a relatively constant amount of time despite their sizable hash rate. Therefore, this scheme can eliminate the fluctuation of the block chain hash rate caused by DAA.

2.4 Proofs of Data Replication and Retrievability

PORs and PDPs that support storing additional replicas of files require the client to process, structure and upload replicas of their files by themselves. This step brings additional bandwidth overhead to both the server and the client, and also brings new security risks to the server [15]. Specifically, since uploaded files are usually encrypted, the server cannot identify whether the uploaded content is a replica of the particular file. This situation limits the business models available on the server-side. For example, the price charged for storing a replica is often lower than the price for storing the original file, which can be abused by malicious users who upload non-replica, even though they claim to be a replica. Armknecht et al. propose a solution that conforms to the current cloud model to Proofs Data Replication and Retrievability (PORR) in the cloud, called Mirror [15]. Mirror utilizes a tunable replication scheme. The scheme is based on a combination of Linear Feedback Shift Registers (LFSRs) with the RSA-based puzzles of Rivest [31]. By doing so, Mirror shifts the burden of building replicas onto the server, so server are likely to appreciate it because it allows server to trade relatively cheap computing resources for expensive bandwidth resources. We will mainly compare our experimental results with Mirrors in Chapter 5.

In a recent study, Guo et al. claimed that Mirror is vulnerable to substitution and forgery attacks [32], creating new security risks for cloud users. The attacks are described in detail as follows:

- *Storage saving attack*: The attack is made possible because all sectors of each challenged replicated block are appended to their corresponding labels in the form of products. The rational server can only store

products of all sectors of each replicated block rather than the data sector itself, thus greatly saving storage resources. Therefore, the rational cloud provider only invests less storage cost compared to the honest cloud provider, however, the client will not notice such behaviour, and the verification result is still showing the rational cloud provider stores the data correctly. In this case, Mirror is no longer safe [32].

- *Substitution attack*: The authors of Mirror claim that if the challenge parameter R = 0 (R indicat which replicas will be involved in the challenge), only Proof of Retrievability (POR) is performed without checking the replica. In this case, however, rational cloud providers can replace some of the challenged blocks (possibly deleted) with other intact blocks that the client cannot detect due to a lack of block index authentication [32].

- *Forgery attack*: Given all valid block-tag pairs, a rational cloud provider can easily forge another valid block-tag pair without the client's secret key by multiplying some of these blocks with their corresponding tags. The rational cloud provider can then replace any block-tag pairs in the original file with a fake pair, compromising the integrity of the original file. However, the client cannot detect the misbehaviour [32].

To sum up, the first attack defeats the security goal of storage allocation, and the latter two attacks defeat the security goal of retrievability. In response to these problems, Guo et al. developed an Improved Proofs of Retrievability and Replication (IPOR2) scheme to overcome security issues in Mirror [32]. The idea to deal with the storage saving attack is to randomly sample a non-empty appropriate subset S of s sectors for each replicated block involved, it indicates which sectors of each involved replica block will be checked in the challenge. So rational cloud providers cannot pass the challenge just by storing the product value of each replica. The authors in [32] prove that the number of checked sectors $|S|$ is set to s - 1 is an optimal choice, which imposes a significant storage overhead on the server while achieving the maximum probability of misbehaviour detection. Guo et al. prevent substitution attacks

and forgery attacks by designing a secure authentication tag by binding block index information.

In 2020, Gritti proposed a publicly verifiable PORR by combining the POR scheme and the exponential-based VDF scheme in a finite group [16]. This scheme retains the characteristics of VDF, the puzzle evaluation is slow, but it is easy and fast to verify the evaluation output. It thus ensures that the server stores both the original file and its replicas at rest, rather than dynamically computing them when requested to prove correct storage. The two PORR schemes mentioned above, Mirror and IPOR2, only support private authentication. That is, only the client can challenge the server. Gritti's PORR scheme allows anyone to challenge the cloud provider, not just the client. As far as we know, this is the first PORR scheme that offers public verification.

2.5 Summary

PDP is the first proposed scheme to verify whether untrusted servers store client data. Later, it gradually expanded the client data replicas and seamless access to dynamic data. However, some schemes still have some problems of high I/O and computational overhead [8, 12, 14]. Later POR schemes not only focus on the integrity of data and replicas, but more research focuses on ensuring the data recovery. Recent PORR schemes combine the characteristics of PDP and POR and shift the construction of replicas burden to the cloud provider, providing the convenience of use for the client, and it allows the server to trade relatively cheap computing resources from expensive bandwidth resources, which also brings economic benefits to the cloud provider. PORR with public verification can greatly improve the applicability of the scheme [16].

Chapter III

Definition

This chapter introduces our implementation steps of Gritti's P-PORR scheme. In general, it can be divided into two phases, the storage phase and the verification phase. The storage phase includes protocol setup and file replication; the verification phase includes challenge generation, response generation and response verification. The specific workflow refers to Figure 3.1. We describe in detail what roles the client and server will play in each phase, what parameters are required for each phase, and the parameters returned.

Symbol	**Meaning**
p, q	Prime numbers used for RSA Setup
N	RSA modulus
d	RSA private key
e	RSA public key
G	Full-domain hash function
H	Full-domain hash function
$\mathbb{Z}_N$	$\{0, 1, 2, \cdots, N-1\}$
$\mathbb{Z}_N^*$	$\mathbb{Z}_N \backslash \{0\}$
t	VDF delay parameter
B	VDF security parameter
L	First t odd prime numbers

P	The product of all elements of L
w	$\{1, 2, \cdots, B\}$
M	Original file, $\{0,1\}^*$
n	Number of blocks
s	Number of sectors
r	Number of replicas
i	Block index
j	Sector index
k	Replica index
$m_{i,j}$	Sector
T	File tag
σ_i	File authenticators for each block
M^*	Processed file
$x_{i,j}^{(k)}$	Puzzle
$y_{i,j}^{(k)}$	Solution of puzzle
$L_{i,j}^{(k)}$	Mapping to puzzles of size κ
$S_{i,j}^{(k)}$	Mapping to puzzles of size κ
$P_{i,j}^{(k)}$	Product of $L_{i,j}^{(k)}$
$g_{i,j}^{(k)}$	Product of g_w for $w \in S_{i,j}^{(k)}$
$m_{i,j}^{(k)}$	k-th replica of $m_{i,j}$
I	A set in $[1, n]$ of l elements
v_i	Random elements $\in \mathbb{Z}_N$ for $i \in I$
(Q, R)	Challenge

(μ_j, σ)	Response
l	Number of challenged blocks

Table 3.1: Symbols and meanings

3.1 Protocol Setup

This phase first initializes the POR and VDF protocols and generates the required keys. Then it splits the file into blocks and sectors, finally encrypts and encodes them with an erasure code. All steps in this phase will be implemented by the client.

We use (RSA.KeyGen, RSA.Sign, RSA.Verify) as an RSA digital signature scheme. Let $\kappa = (\kappa_1, \kappa_2)$ be the security parameter. We choose two random prime numbers p and q, they satisfy that $p, q \in [2^{\kappa_1 - 1}, 2^{\kappa_1} - 1]$. Let $N = pq$ be the RSA modulus such that $2^{\kappa_1 - 2} < N < 2^{\kappa_1}$ and the bit length of N is the modulus size. We randomly pick a prime number e of length $2\kappa_1 + \kappa_2$ bits as the RSA public key, and $d = e^{-1} \mod \phi(N)$ as the RSA private key [16].

Let $G : \{0,1\}^* \to \mathbb{Z}_N^*$ and $H : \mathbb{Z} \to \mathbb{Z}_N$ be full-domain hash functions, seen as random oracles. The $\mathbb{Z}$ is a set of integers, $\mathbb{Z}_N$ is the set of $\{0, 1, 2, \cdots, N-1\}$, and $\mathbb{Z}_N^*$ is $\mathbb{Z}_N \backslash \{0\}$ [16].

We set t be the delay parameter and B be the security parameter for VDFs. Let $L = \{l_1 = 3, l_2 = 5, \cdots, l_t\}$ be the first t odd prime numbers and $P = l_1 \times l_2 \times \cdots \times l_t$, the parameter P is a large integer with about $t \log t$ bits. The client computes $h_w = H(w)$ and $g_w = h_w^{1/P}$ for $w \in [1, B]$ [16].

The client executes RSA.KeyGen(κ) to generate the signing and verification key pair (ssk, spk).

We assume the file that is uploaded to the server is $M \in \{0,1\}^*$. File M is encrypted and encoded with erasure codes. The encryption of files ensures sufficient privacy and effectively avoids man-in-the-middle attacks, even cloud

service providers have no right to know the contents of files. Encoding guarantees the file extractability and the recovery of the entire specific file [10]. The file M is split into n blocks, and further into s sectors. Each sector is denoted as $m_{i,j} \in \mathbb{Z}_N$, for $i \in [1, n]$ and $j \in [1, s]$. In order to achieve an unique extractability [15], the bit representation of each sector $m_{i,j}$ is supposed to contain a characteristic pattern, such as a sequence of zero bits. The pattern length depends on the size of the file and should be larger than $\log_2(n \cdot s)$ [16].

The client also generates file tag and authenticators. Each file has a unique file tag and n authenticators (depending on the number of blocks). First, the client chooses a random file identifier $id \in \mathbb{Z}_N$. Then client picks random s non-zero elements $u_1, u_2, \cdots, u_s \in \mathbb{Z}_N$. The client computes $T_0 = id||n||u_1||u_2||\cdots||u_s$, finally the file tag is $T = T_0||\mathsf{RSA.Sign}_{ssk}(T_0)$. For the file authenticators σ_i, the client calculates $\sigma_i = (G(id||i) \cdot \prod_{j=1}^{s} u_j^{m_{i,j}})^d \mod N$ for $i \in [1, n]$. All operations above are done in the multiplicative group $\mathbb{Z}_N^*$ of invertible integers modulo N, this rule is working for the following phase as well [16].

We assume that the client requires the server to generate r replicas of the original file M. In this case, the puzzles for VDF are denoted as $x_{i,j}^{(k)}$ for $i \in [1, n]$, $j \in [1, s]$ and $k \in [1, r]$. The client uploads the processed file $M^* = (\{m_{i,j}\}_{i\in[1,n],j\in[1,s]}, \{\sigma_i\}_{i\in[1,n]})$ to the server [16].

In summary, the public parameters are $\{N, \kappa, L, B, P, \{g_w\}_{w\in[1,B]}, \{x_{i,j}^{(k)}\}_{i\in[1,n],j\in[1,s],k\in[1,r]}\}$. They are shared between client, server and any parities who are interested to be a verifier. In addition, the verification parameters are $\{T, spk, e, G, H\}$, they are shared between client and verifier. In the end, the client keeps a set of private parameters $\{ssk, d\}$. There are more symbols that will appear in the subsequent phases. All symbols and meanings are shown in Table 3.1 later.

3.2 File Replication

As what we mentioned before, the burden of building replicas is shifted onto the cloud provider, so after the server receives the processed file M^* and public parameters, it starts generating the agreed-upon r replicas. The server needs

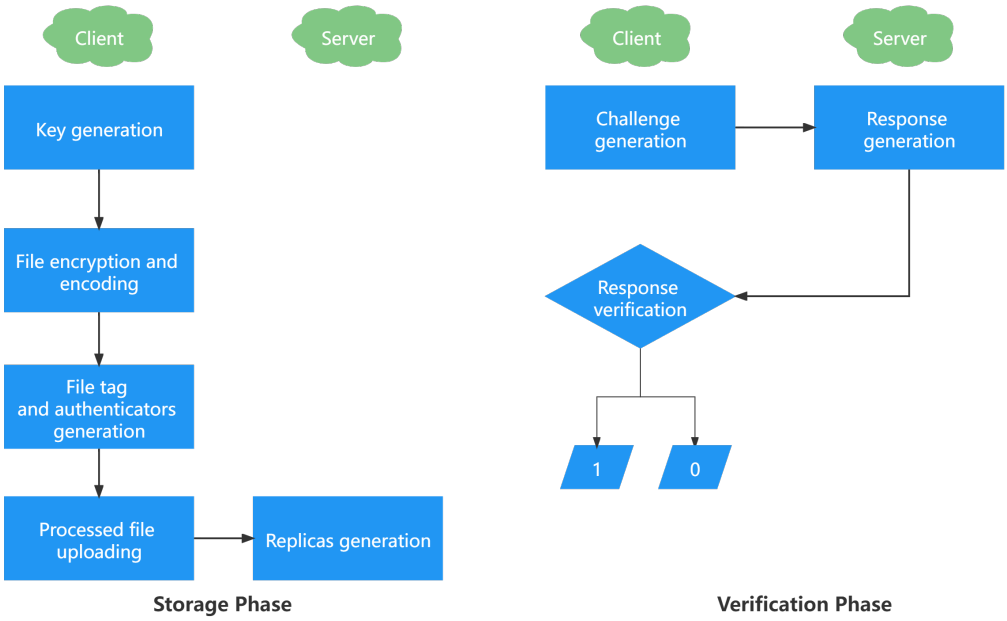


Figure 3.1: PORR protocol workflow diagram

to evaluate the VDF puzzles $x_{i,j}^{(k)}$ and get the corresponding solutions $y_{i,j}^{(k)}$. The server firstly uses a random hash function to map the $x_{i,j}^{(k)}$ to $L_{i,j}^{(k)} \subseteq L$ of size κ and the random subset $S_{i,j}^{(k)}$ of κ values in $[1, B]$. The server secondly computes the product $P_{i,j}^{(k)}$ of all primes in $L_{i,j}^{(k)}$ and $g_{i,j}^{(k)} = \prod_{w \in S_{i,j}^{(k)}} g_w$. Then the server can computes $y_{i,j}^{(k)} = (g_{i,j}^{(k)})^{P/P_{i,j}^{(k)}} \in \mathbb{Z}_N$. Finally, the server can build the replicas $m_{i,j}^{(k)}$ of the original sectors $m_{i,j}$ for $k \in [1, r]$, it is done by computing $m_{i,j}^{(k)} = m_{i,j} + y_{i,j}^{(k)}$ [16].

The server has now completed the generation of the replicas, and it will also retain the puzzle solutions for future challenges. The above steps are assumed to be the behaviours of an honest server, and a rational server that does not take these steps will most likely not be able to respond to the challenge appropriately. Indeed the rational server needs to calculate the solutions of all puzzles in the required data block on the fly after receiving the challenge, this will take some easily observable and unreasonable time.

3.3 Challenge Generation

Gritti's PORR [16] supports public verification, therefore the client and any authorized verifiers can generate a challenge to the server. If the verifier is not the client, it will be shared with the verification parameters $\{T, spk, e, G, H\}$ to complete the challenge generation.

Due to security risk, we need the verifier to check the signature first to confirm that the data sent from the client have not been tampered during transit [33]. The file tag is given by $T = T_0||\mathsf{RSA.Sign}_{ssk}(T_0)$. The verifier firstly executes $\mathsf{RSA.Verify}_{spk}(T_0, \mathsf{RSA.Sign}_{ssk}(T_0))$ to check if the signature is valid, if not, the verifier halts the process; if so, the verifier recovers the elements $id||n||u_1||u_2||\cdots||u_s$ from the file tag. Then the verifier randomly chooses a block index set $I \subset [1, n]$ of l elements, it denotes that the verifier is about to send one challenge on l blocks, with blocks' indices in the set I. The verifier also randomly picks $v_i \in \mathbb{Z}_N$, for $i \in I$, and sets $Q = \{(i, v_i)\}_{i \in I}$ where i corresponds to the index of a block m_i. The verifier finally chooses a random set $R \subset [1, r]$ as the replica indexes must be checked. The challenge is set as $chal = (Q, R)$ and sent it to the server.

3.4 Response Generation

Once the server receives the $chal = (Q, R)$ from verifier, the server starts to generate the response $resp$ for the challenge. The server firstly computes $\mu_j = \sum_{(i,v_i)\in Q} v_i m_{i,j} \in \mathbb{Z}$ and $\sigma = \prod_{(i,v_i)\in Q}(\sigma_i \cdot \prod_{j=1}^{s} \prod_{k\in R} u_j^{m_{i,j}^{(k)}})^{v_i} \mod N$. The response $resp$ is set as $resp = (\{\mu_j\}_{j\in[1,s]}, \sigma)$ and send it back to the verifier. If a rational cloud provider runs the server, it might compute the puzzle solution y at this phase to respond to the challenge correctly. But since the VDF requires a specified number of sequential steps to evaluate the puzzles [27], even if the server uses powerful hardware for parallel computing, it cannot significantly reduce computing time to cover up the misbehaviour.

3.5 Response Verification

This phase is important, it theoretically only takes a short time to complete, and finally confirms whether the behavior of the server is genuine. After the

verifier receives $resp = (\{\mu_j\}_{j\in[1,s]}, \sigma)$ from server, the verifier first checks if μ_j satisfies that $0 \leq \mu_j \leq l \cdot N \cdot (N-1)$ for all $j \in [s]$. If any values are not in the range, the verifier treats this response $resp$ as failure and outputs 0. Otherwise the verifier computes $P_{i,j}^{(k)}$ and $S_{i,j}^{(k)}$ for each puzzle (include in the challenged blocks) $x_{i,j}^{(k)}$ for $i \in I$, $j \in [s]$ and $k \in [R]$, also $h_{i,j}^{(k)} = \prod_{w \in S_{i,j}^{(k)}} H(w)$ [16].

In the end, the verifier checks whether the following equation holds, if so, the verifier outputs 1; otherwise 0. The correctness of the protocol has been proved in Appendix [16].

$$\sigma^e = \prod_{(i,v_i)\in Q} G(id||i)^{v_i} \times \prod_{j=1}^{s} u_j^{\mu_j(1+|R|e)} \times \Big(\prod_{(i,v_i)\in Q} \prod_{j=1}^{s} \prod_{k\in R} u_j^{v_i (h_{i,j}^{(k)})^{1/P_{i,j}^{(k)}}} \Big)^e \mod N$$

Chapter IV

Experiment Design

This chapter first describes our overall experimental environment and the configurations of the test host, then introduces the default parameter settings of P-PORR in the experiments, finally explains how to simulate a realistic Wide Area Network (WAN) environment in the experiments.

4.1 Implementation

In order to comprehensively analyze the performance of P-PORR and the different behaviours of honest and rational cloud providers, we design and implement a P-PORR scheme. Most of our experimental settings follow a similar work [15] for more accurate and meaningful comparisons. Our code is written in Python 3.8 and we rely on a hash function SHA-256 and PyCryptodome's built-in random number generator. We used the Python library Zfec to implement an erasure code, it can be parameterized to choose in advance the number of elements whose loss it can tolerate. The proportion between redundant blocks and data blocks was set to third, which means that as long as the lost data blocks do not exceed one-third of the total data blocks (redundant and data blocks), it is recoverable.

We deployed the entire test environment on a PC running Intel Core i7-9700 with 32GB Random Access Memory (RAM). Four Virtual Machines (VMs) were created on this PC to design our test environment. The specific structure topology is like Figure 4.1:

- One VM represents the client that owns the files stored on the cloud storage provider (server), and it has two-way communications with the server, including uploads and downloads. It also represents verifiers that are allowed to generate challenges to the server.

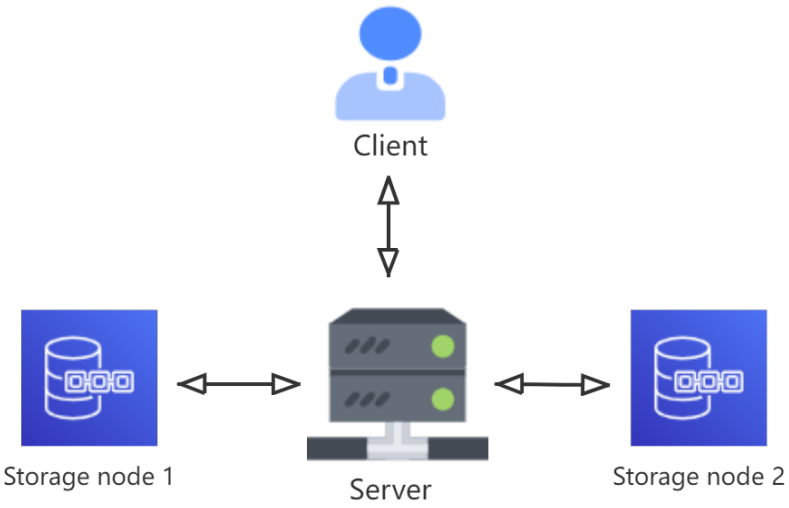


Figure 4.1: Experimental environment topology

- One VM represents a cloud storage provider (server) that provides cloud storage services, and it uploads and downloads data on storage nodes according to client's needs.

- Two VMs representing two storage nodes, they connect to the cloud storage provider and store the client's data at rest.

4.2 Parameter Selections

For more meaningful comparisons with existing similar works, our default parameter selections are similar to [15]. As the core algorithm in P-PORR, the setting of the RSA key will significantly affect the performance of P-PORR due its size. The early recommend modulus size of RSA was 512 [34], the specific evaluation of its security at that time showed that the cracking cost might not be lower than $1,000,000 and 8 months of effort [34]. With the rapid development of computer hardware, the cracking efficiency of hackers has significantly improved. The latest research shows that the recommended key size is 1024 bits for personal use and 2048 bits for companies use [35, 36, 37].

Parameter	**Default value**
RSA modulus size $\|N\|$	2048 bits
RSA prime size $\|p\|$	1024 bits
RSA prime size $\|q\|$	1024 bits
Pre-processing parameter B	2^{30}
Delay parameter t	2^{16}
File size	64MB
Block size	8192 Bytes
Sector size	256 Bytes
Number of replicas r	2
Number of challenges $\|I\|$	40

Table 4.1: Default parameter selections in the experiment environment

A giant security key will bring more security. However, it will also affect the user experience in encryption/decryption time, power consumption and memory usage, so we finally chose 2048-bit RSA modulus size and 1024-bit prime numbers p and q.

The pre-processing parameter B is used to ensure that the VDF is safe against bounded pre-computing attacks. Our VDF scheme is secure for B challenges as long as the adversary cannot run a long pre-computation during a period, between the time that the public parameters are made public and the time when the VDF is evaluated, after which new public parameters need to be generated [27, 16]. The recommended setting for B is 2^{30} in [27], in which case a client can store up to 2000 files using one VDF setting, or about 128GB size of files [16]. The free storage size provided by Amazon AWS is 100GB [38], which can be understood as the basic needs of ordinary clients. The default file size in our experiment is 64MB (the file size will change in different experiments). The file size of 128GB is larger than both previous situations, so 2^{30} for B is acceptable. Delay parameter t defines that an honest cloud provider can evaluate a puzzle in t sequential steps. At the same time, a parallel machine adversary with a polynomial number of processors cannot

distinguish the output y from random in a smaller number of steps. We set the delay parameter t equal to 2^{16} [27].

A block size of 8KB provides the most balanced average performance in cloud storage [15, 39], so we set the default block size to 8KB. A file with a default size of 64MB was split into 8000 blocks; each block is divided into 32 sectors, each sector's size is 256 Bytes, and each sector corresponds to a puzzle. The default number of replicas is 2, so the server needs to evaluate 8000 * 32 * 2 puzzles. The default number of challenges is 40, which also means that each challenge will randomly select 40 blocks of data for verification.

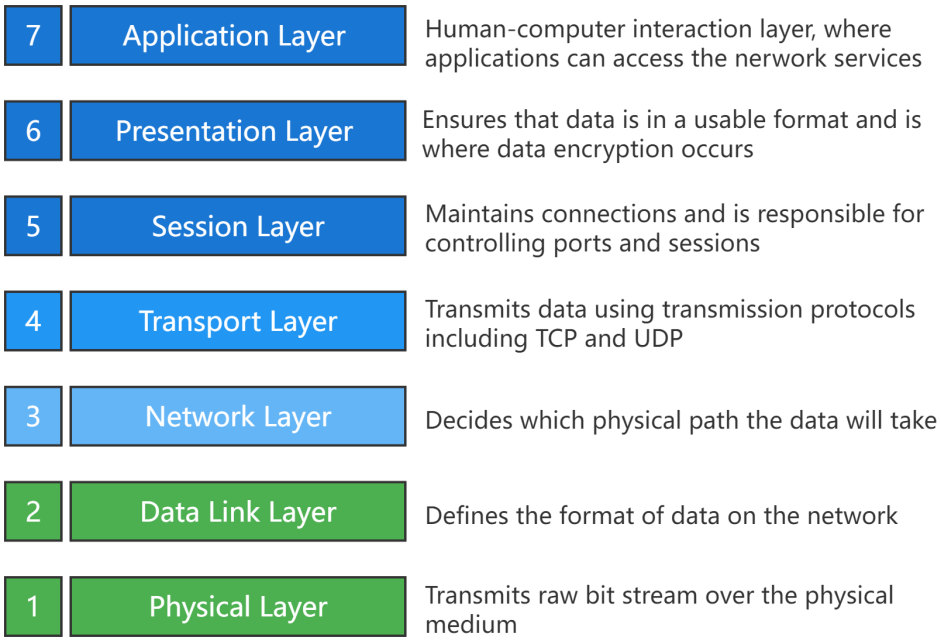


Figure 4.2: Open Systems Interconnection (OSI) 7-layer model[1]

We tested three commonly used network storage and sharing protocols on the already built environment .

- File Transfer Protocol (FTP): The FTP works at the Application Layer in the Open Systems Interconnection Model (OSI) model, shown in Fig-

[1] https://www.imperva.com/learn/application-security/osi-model/

ure 4.2. The client needs to use a particular FTP client to communicate with the server. It can work on any operating system and protect our data by asking the user name and password as login credentials, but by default FTP will transfer the credentials unencrypted [40, 41].

- Server Message Block (SMB): It is mainly used but not limited to computers running Microsoft Windows. It also provides an authenticated Inter-Process Communication (IPC) mechanism. The client does not need any technical knowledge to share folders and download files. However, because of its security vulnerabilities, it will be used by hackers as a backdoor attack [42, 43].

- Network File System (NFS): It is mainly used but not limited to computers running Unix or Unix-like computers (such as Linux). The implementation of NFS uses the Remote Procedure Call (RPC) mechanism, so that the client can call the function of the server. The operating system kernel sends the call request of the NFS file system to the NFS service of the server through TCP/IP, and related operations are executed. The server then returns the result of the operation to the client [41].

We tested the upload speed of files of different sizes under the three protocols, and the experimental results are shown in the Figure 4.3. Due to the instability of network transmission, the upload speed results show inevitable and acceptable fluctuations. Nevertheless, it can still be seen that the FTP has the fastest speed of these three protocols. The average upload speed of FTP is 17.4 MB per second, which is about 1 second faster than NFS and about 3 seconds faster than SMB. It also has stable security. Therefore, FTP is selected as the default transmission protocol in our experiments.

4.3 Network Environment Design

Network emulators are widely-used test tools in developing network protocols and applications. An important reason for their popularity is their ability to

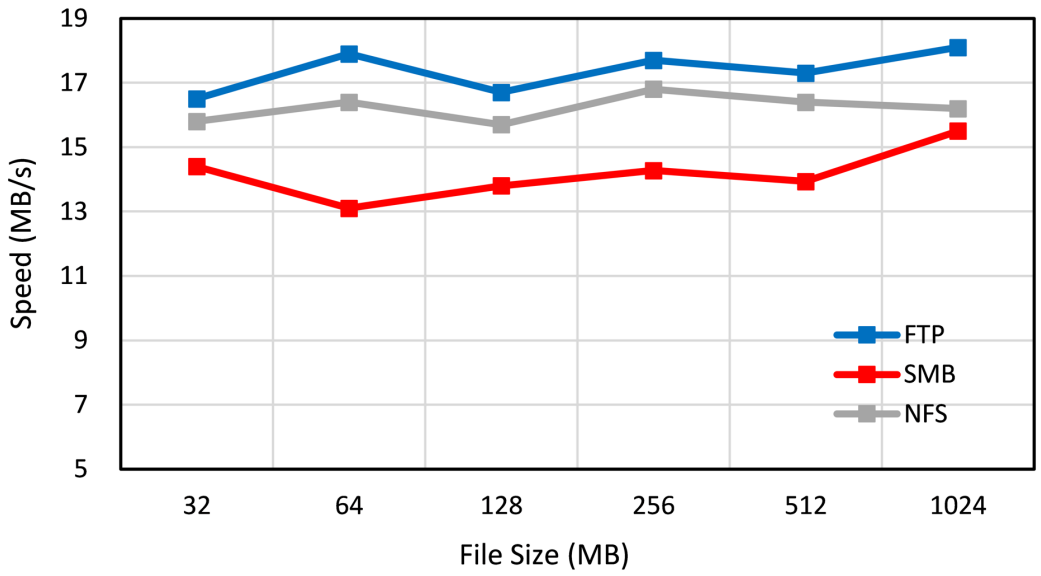


Figure 4.3: Network storage and sharing protocols.

perform tests under tightly controlled network conditions that would be difficult to achieve in real networks. Network emulation can be implemented using specialized hardware or through software, usually requiring support from the operating system kernel [44, 45]. Software-based emulators can provide a high accuracy regardless of network traffic load and are more accessible. The NetEm is a popular version of software-based emulators that is part of Linux kernel 2.6 and above [46]. It provides the ability to accurately simulate most network parameters such as simulating packet delay, packet loss, duplication and re-ordering through the Linux Traffic Control (tc) tools, and utilizes Token Bucket Filters (tbf) to simulate rate control [44].

To simulate a real Wide Area Network (WAN), the communication between the VMs is bridged using a 100 Mbps switch. All traffic exchanged in the environment is shaped by NetEm [47], and the specific network parameters are set as follows:

- We set the packet loss rate at 0.1%, and the correlation rateof lost packets at 0.001% [48, 49, 50].

- We set the delay rate to fit normal distribution with a mean of 20ms

and a variance of 4ms [51].

- We set the packet corruption rate at 0.1% [52].
- We set the rate of the package being out of order at 0.2% [53].

Chapter V

Result Analysis

In this chapter, we analyze the measurement results obtained under the experimental configuration in the previous chapter from multiple perspectives. It includes the time consumption of P-PORR at different stages, how different block and sector sizes affect P-PORR's performance, and the financial impact on cloud service providers. The experiment results prove that the PORR protocol from [16] generates fair computing and communication overhead on the client, cloud provider, and verifier, which means that this solution is practically applicable in a cloud storage environment. Each data point in our following plots is averaged by 5 independent measurements; where appropriate, we also show the corresponding 95% confidence intervals.

5.1 Performance at File Preparation

The file preparation time is showing the total time spent on a file before storing it on the cloud. It includes the time spent on key generation, tag generation and puzzle evaluation. Results are shown in Figure 5.1. When creating two replicas of a 64MB file, the overall preparation time in our case is about 180 seconds. The plot of the Mirror shows that it will take about 727 seconds to replicate two replicas of a 64MB file [15], P-PORR seems to show a significant advantage. We conducted a further comparison. When Mirror creates eight additional replicas for a 64MB file, the total file size that needs to be processed is 512MB, which is the same as the 256MB file with two replicas in our case. With this parameter configuration, Mirror only consumed about 765 seconds [15], while our situation consumed about 723 seconds. P-PORR still maintains a weak advantage, but it can be seen that the increase in P-PORR file preparation time has become evident as the file size increases. While the increase in Mirror is not much, it replicates six

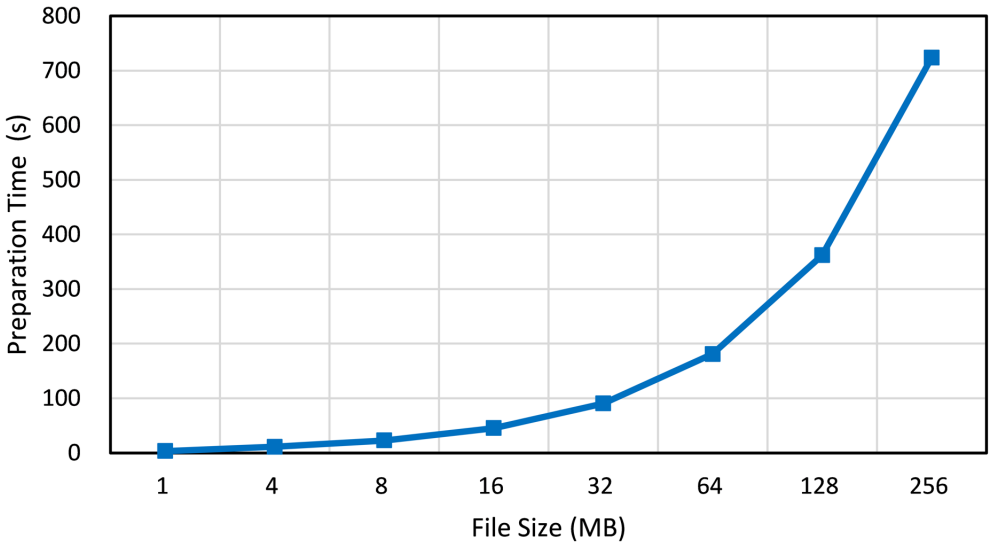


Figure 5.1: File preparation time

additional files only adds 38 seconds of expense [15].

Once the number of additional replicas exceeds eight, the Mirror replication time rise begins to be exponential increase. It causes a delay of approximately 1250 seconds when 16 additional file replicas are created [15]. This situation is because Mirror uses multiple threads for replication, which can be executed in parallel. However, with the increase in the number of concurrent replication requests, the eight threads in the thread pool have been exhausted, and the system will be saturated. It took 2887 seconds to prepare files of the same size in our case, which remained the same as the previous trend.

5.2 Performance at Verification

Clients' most frequently used service is to verify along with the original file are correctly stored at rest. To this end, we measured the total time required for the server to generate the response and the client to verify the response. A bigger number of challenges will increase the probability that a rational server's malicious behaviour will be detected. Thus, the number of challenges is variable; it is calculated as half of the file size (MB). For example, 32

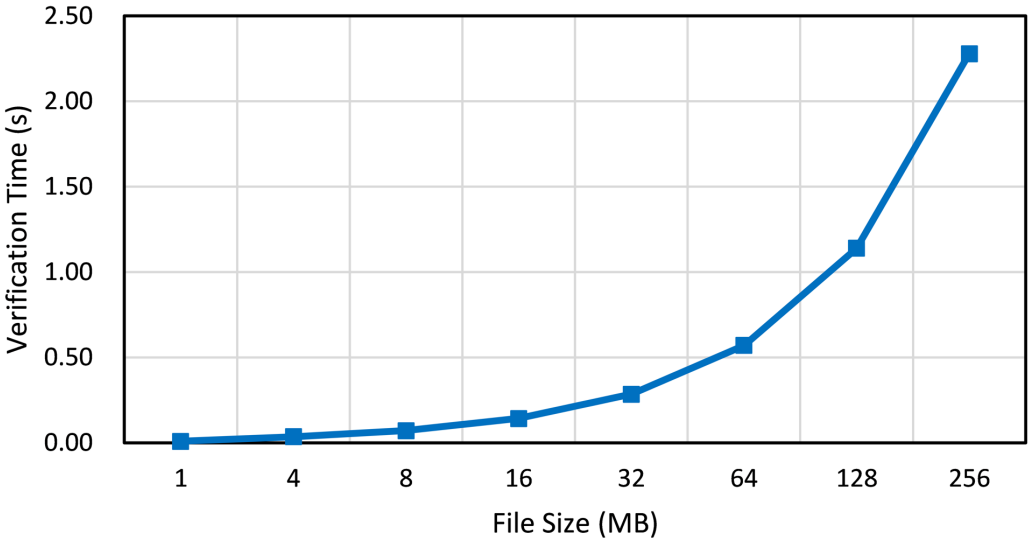


Figure 5.2: Verification time (verifier)

challenges are requested for a 64MB file. According to the default parameter settings, the verifier thus checks 128KB of data for each MB of file data. The experimental results are shown in Figure 5.2.

In our case, when the file size is 64MB, the number of challenges initiated is 32, which is roughly the number of challenges in Mirror (set to 40) in Mirror [15]. The verification time seen by the client in our case is 0.57 seconds; it is 0.8 seconds in Mirror [15]. As our file size increases, the number of corresponding challenges increases. For example, when the file size is 128MB, the number of challenges is 64, and the verification time rises to 1.14 seconds. The number of challenges increases the latency for client, but it ensures stable security. In Mirror's experiment, the number of challenges is fixed at 40, independent of the file size. It is the main reason why the verification time remains 0.8 seconds. It provides an instant verification speed, but the security risk of large files becomes high with that.

5.3 Performance at Puzzle Evaluation

The puzzles directly restrict the misbehaviours of the server. Preparing the puzzles takes the longest time in file preparation. It contributes about 72% of

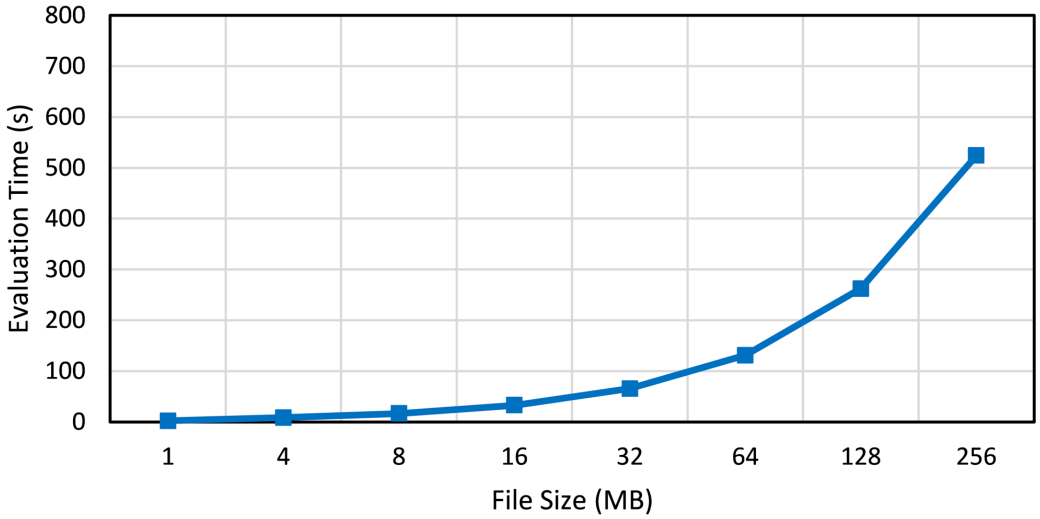


Figure 5.3: Puzzle evaluation time

the file preparation. Thus, we measure the time spent for puzzle evaluation only. The experimental results are shown in Figure 5.3. In the default configuration, a 64MB file requires 180 seconds of file preparation time, of which 131 seconds are spent on puzzles evaluation. As the file size increases, the number of sectors also increases, which leads to the continuous linear growth of puzzle evaluation.

5.4 Performance at Response Generation

We measure the time required for the server to generate a response based on a different number of challenges. The experimental results are shown in Figure 5.4. We consider the behaviours of rational cloud providers and honest cloud providers. The former requires more time to generate a response because it requires the dynamic calculation of puzzle solutions. On the other hand, since honest cloud providers have calculated solutions to puzzles and stored them in a static state, their response generation speed is significantly faster.

In our case, under the same number of challenges, the response generation time of a rational server is roughly twice that of an honest server. The response generation time continues to rise as the number of challenges increases.

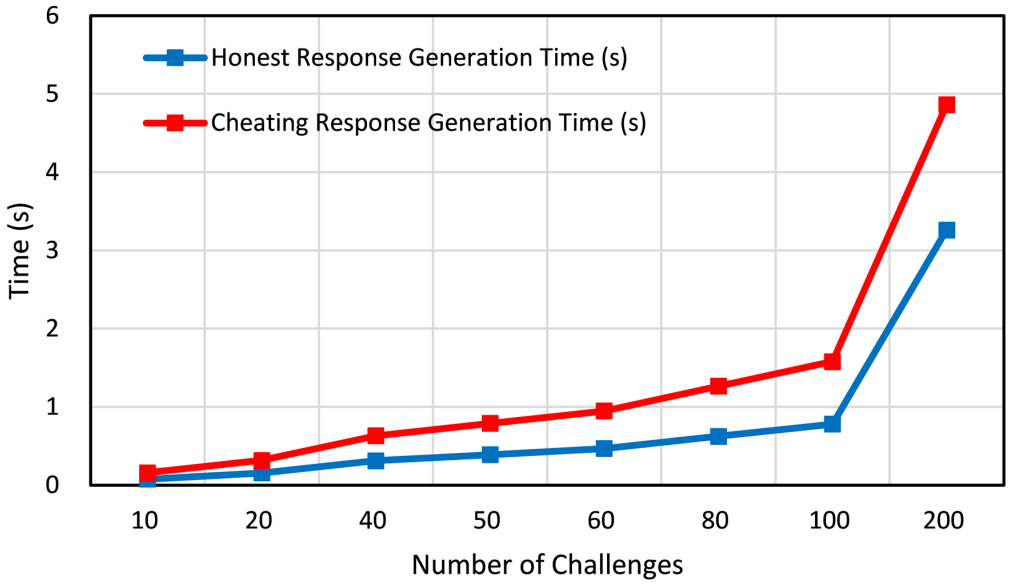


Figure 5.4: Response generation time

The response generation time given in Mirror is based on one (bitlength of coefficients) of the blind factors, which is different from our analysis of response generation time based on the number of challenges. However, the authors of Mirror claim that as long as the blind factors are greater or equal to 70, the rational server should not gain any (reasonable) advantage in misbehaving. According to that, we know that when the bitlength of coefficients is equal to 70, the rational server in Mirror takes about 1 second to generate the response of 40 challenges [15]. While the rational server in our case took 0.63 seconds to generate the response of 40 challenges, at this time, the honest server took only 0.31 seconds. In this comparison, the number of challenges in Mirror and our case are both 40. Our case has the advantage of response speed in this context.

If we observe the difference between Mirror and P-PORR at the highest security level, we will find that the rational server in Mirror only takes about 2.8 seconds [15]. In our case, the rational server reached 4.86 seconds. In this situation, we launched 200 challenges for a 64MB file. The rational server needs to calculate more puzzle solutions on the fly, and Mirror still launches 40 challenges. The security method Mirror uses in order to increase

the computational cost of a rational server is the increase of the bitlength of the blind factors.

5.5 Performance at Different Block Sizes

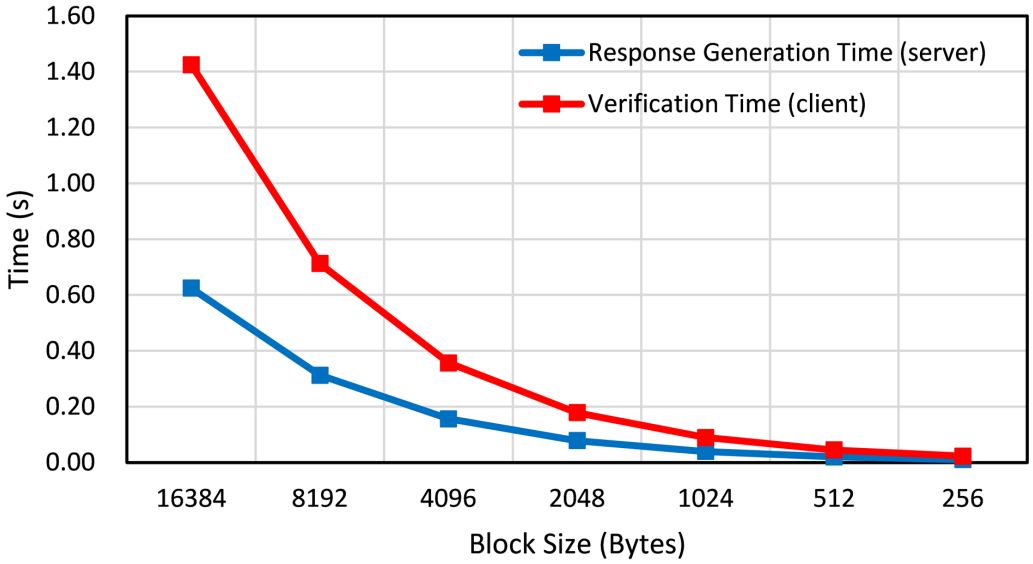


Figure 5.5: Latency at different block sizes

According to the block sizes, we tested the time required for response generation and verification. The default sector size is 256 Bytes. The experimental results are shown in Figure 5.5. We notice that a smaller block size gives a shorter time. Indeed, block size and challenge number are related, meaning that response generation and verification are faster when the block size decreases.

There is a related plot in Mirror showing that it has the same trend as ours [15], but our trend is more obvious. When the block size in Mirror is 2048 Bytes, the client verification time is about 1.1 seconds, compared to 1 second at 1024 Bytes hence a difference of 0.1 seconds. In our case, the verification time is around 0.1 seconds for a 1024 Bytes block size, when the block size is 2048 Bytes, the time increased by 50%, which is about 0.2 seconds. This is still more optimistic than [15]. The previous comparison also shows that our situation has more advantages in handling small files.

5.6 *Performance at Different Sector Sizes*

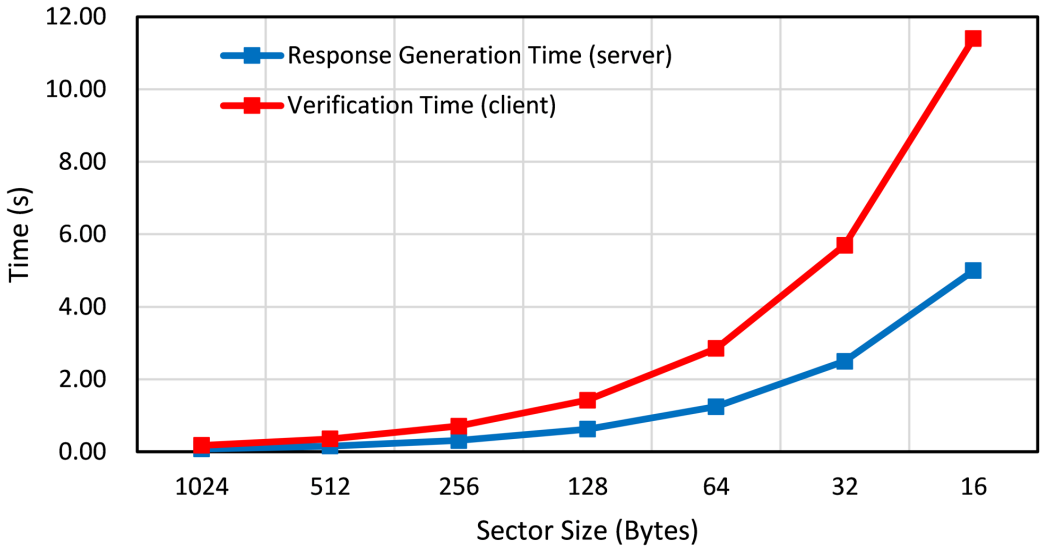


Figure 5.6: Latency at different sector sizes

We tested the time required for response generation and verification according to the sector sizes. The default block size in the experiment is 8192 Bytes. The smaller sector size means more number of sectors for a given block size. The number of blocks and challenges are fixed as the default setting. It affects the verification time since the verifier must check more data from the challenges. The experimental results are shown in Figure 5.6.

Mirror does not explicitly analyze the effect of sector size on its protocol, so our discussion is mainly based on our case. In general, when the sector size decreases, the server response time and client verification time will increase. This will also significantly improve security because under the same number of challenges (blocks), a smaller sector size means that more data will be verified. We finally chose 256 Bytes as the default sector size to balance verification time and security. It only takes 0.36 seconds for the client to get the verification result. At the same time, the sector size of 32 Bytes will take 15 times slower than the default setting on verification.

5.7 *Financial Considerations*

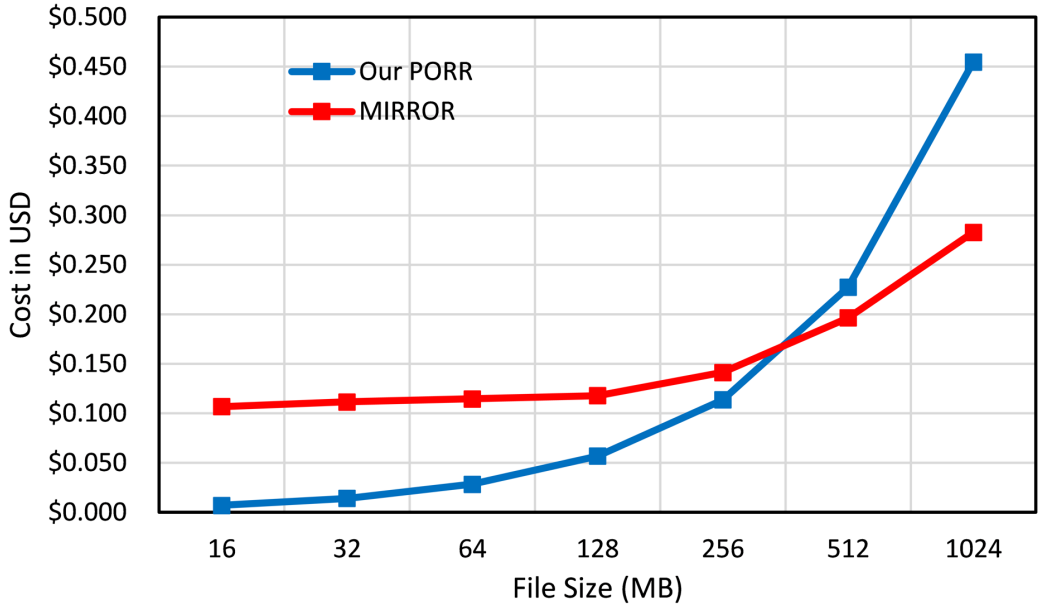


Figure 5.7: Financial costs on cloud service providers

The financial benefit is an aspect that cloud service providers attach importance to. We calculated the financial costs of P-PORR and Mirror in preparing files. The Mirror data comes from the replicating process, which is the same as the file preparation time in our case. The experimental results are shown in Figure 5.7. We assume that the server provides a large general-purpose instance from Amazon EC2 at US$0.404 per hour, equivalent to US$0.000112 per second and multiplied by our file preparation time (in seconds).

When the file size is 384MB, both P-PORR and Mirror will incur costs of approximately US$0.18 [15]. From a financial point of view, our case is more competitive when processing files smaller than 384MB, which can save approximately 2 to 15 times compared to [15]. Once the file size exceeds 384MB, Mirror gradually gains an advantage due to its slow increase when preparing files. Most of the previous technical results show that our case is competitive in handling small files, which also maps to finances.

Chapter VI

Conclusion

In this chapter, we discuss our main findings and their implications. We also examine limitations in our work and future research directions.

6.1 Discussion

The experimental results in Chapter 4 show that P-PORR exhibits acceptable results in realistic cloud environments. Rational cloud providers take a significantly longer time to respond to verification than honest cloud providers, demonstrating that the combined usage of VDF and POR is successful and can be applied to proofs of data replication and retrievability in the cloud. Achieving a fast response verification time is also consistent with our initial assumptions when combining VDF with POR. The experimental results show the similar performance of P-PORR and related research at this stage in many aspects, which proves the practicability of P-PORR. Especially when dealing with small files, whether it is the time for the server to prepare the file, the time for the server to generate the response, the time for the client to verify the response, or financial considerations, P-PORR all showed better performance than similar solutions at this stage, such as Mirror. Especially P-PORR is proven to be publicly verifiable, and the verifier only needs a few parameters to complete the challenge-verification process. This feature fully guarantees the client's privacy because the verifier cannot know any file content of the client through these parameters. However, it brings enough convenience to the client because it could help the client who has a weak security awareness to supervise the cloud provider.

This experiment provides new insights into the relationship between VDF and data replication and retrievability proofs in the cloud. VDF was usually used for rate or network traffic filtering in the past. We can focus more

research on VDF on cloud storage proof in the future. As the first implementation of public PORR, the experimental results in Chapter 4 also helps us to clearly understand the difference between public PORR/P-PORR and private PORR/Mirror in usage. They have their characteristics, such as P-PORR is always faster at file preparation time and verification for small files. At the same time, Mirror initially takes longer on multi tests than P-PORR, but its rise is slow and eventually has an advantage in verifying large files.

6.2 Limitations

Any researches have some limitations, and ours is no exception. We deploy the test environment of Chapter 4 in order to make more meaningful comparisons with similar studies. The purpose of our test environment is to simulate the realistic cloud storage and network environment as much as possible, but this is not 100% true due to time constraints. The most popular open-source cloud platform is OpenStack, which uses pooled virtual resources to build and manage private clouds and public clouds. The tools that make up the OpenStack platform are called "projects" and handle core cloud computing services such as compute, networking, storage, identity, and image services [54, 55, 56]. OpenStack is supporting more than 75 other public cloud providers around the world, including well-known companies such as Dell, HP, and IBM [57, 58]. Building and testing P-PORR with OpenStack should yield a more objective experimental result so that the performance of P-PORR can be more comprehensively analyzed from multiple perspectives.

6.3 Conclusion and Future Work

In this paper, we implemented and tested the first public PORR solution P-PORR, which combines a POR scheme and an exponentiation-based VDF scheme [11, 27]. It is used for the client/verifier to check whether the original file and its replicas are stored correctly on the server. We mainly compared the experimental results with a similar private PORR solution, called Mirror [15], and the results show that P-PORR has acceptable performance in a real cloud environment, especially in the replicas generation and verification of small files in terms of time, both are better than the Mirror scheme.

In future work, we will focus on the following points:

- Increasing the retrievability of dynamic data by referring to the Merkle Hash Tree concept.

- Expanding more research on how to increase the robustness of P-PORR in the face of security threats, such as ensuring that the contents of the data packets transmitted between the verifier and the server are not tampered with, and that no one pretends to be a verifier or the server transmitting wrong information to deceive each other.

- Using the OpenStack platform to implement and evaluate the protocol more realistically and comprehensively.

References

[1] Xhemal Zenuni, Jaumin Ajdari, Florije Ismaili, and Bujar Raufi. Cloud storage providers: A comparison review and evaluation. In ., CompSysTech '14, page 272–277, New York, NY, USA, 2014. Association for Computing Machinery.

[2] BBC. Google loses data as lightning strikes. https://www.bbc.com/news/technology-33989384, 2015. Accessed on 11/04/2021.

[3] Medium. Tencent was claimed ten million for data loss due to cloud hard drive glitch. https://medium.com/genaro-network/tencent-was-claimed-ten-million-for-data-loss-due-to-cloud-hard-drive-glitch 2018. Accessed on 11/04/2021.

[4] Computer Weekly. Enterprises exposed to data loss by cloud configuration errors. https://www.computerweekly.com/news/252471175/Enterprises-exposed-to-data-loss-by-cloud-configuration-errors, 2019. Accessed on 11/04/2021.

[5] Amazon. Amazon s3 service level agreement. https://aws.amazon.com/cn/s3/sla/, 2019. Accessed on 11/04/2021.

[6] Yadi Ma, Thyaga Nandagopal, Krishna P. N. Puttaswamy, and Suman Banerjee. An ensemble of replication and erasure codes for cloud file systems. In *2013 Proceedings IEEE INFOCOM*, pages 1276–1284, 2013.

[7] Dell. Protect data stored and shared in public cloud storage. https://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dell_data_protection_cloud_edition__data_sheet_HR.pdf, 2013.

[8] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, page 598–609, New York, NY, USA, 2007. Association for Computing Machinery.

[9] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1), June 2011.

[10] Ari Juels and Burton S. Kaliski. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, page 584–597, New York, NY, USA, 2007. Association for Computing Machinery.

[11] Hovav Shacham and Brent Waters. Compact proofs of retrievability. *J. Cryptol.*, 26(3):442–483, July 2008.

[12] R. Curtmola, O. Khan, R. Burns, and G. Ateniese. Mr-pdp: Multiple-replica provable data possession. In *2008 The 28th International Conference on Distributed Computing Systems*, pages 411–420, 2008.

[13] Ayad F. Barsoum and M. Anwar Hasan. Integrity verification of multiple data copies over untrusted cloud servers. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 829–834, 2012.

[14] Ayad F. Barsoum and M. Anwar Hasan. Provable multicopy dynamic data possession in cloud computing systems. *IEEE Transactions on Information Forensics and Security*, 10(3):485–497, 2015.

[15] Frederik Armknecht, Ludovic Barman, Jens-Matthias Bohli, and Ghassan O. Karame. Mirror: Enabling proofs of data replication and retrievability in the cloud. In *Proceedings of the 25th USENIX Conference*

on Security Symposium, SEC'16, page 1051–1068, USA, 2016. USENIX Association.

[16] Clementine Gritti. Publicly verifiable proofs of data replication and retrievability for cloud storage. In *2020 International Computer Symposium (ICS)*, pages 431–436, 2020.

[17] G.S. Prasad and Vidya Gaikwad. A survey on user awareness of cloud security. *International Journal of Engineering and Technology(UAE)*, 7:131–135, 05 2018.

[18] Cloudflare. The relative cost of bandwidth around the world. `https://blog.cloudflare.com/the-relative-cost-of-bandwidth-around-the-world/`, 2014. Accessed on 11/04/2021.

[19] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, SecureComm '08, New York, NY, USA, 2008. Association for Computing Machinery.

[20] Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, page 109–127, Berlin, Heidelberg, 2009. Springer-Verlag.

[21] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, page 319–333, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 90–107, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[23] Wikipedia. Standard model (cryptography). https://en.wikipedia.org/wiki/Standard_model_(cryptography), 2021. Accessed on 20/01/2022.

[24] Wikipedia. Random oracle. https://en.wikipedia.org/wiki/Random_oracle, 2022. Accessed on 20/01/2022.

[25] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In Michael Backes and Peng Ning, editors, *Computer Security – ESORICS 2009*, pages 355–370, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[26] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, page 325–336, New York, NY, USA, 2013. Association for Computing Machinery.

[27] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[28] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://ia.cr/2018/712.

[29] Vidal Attias, Luigi Vigneri, and Vassil Dimitrov. Preventing denial of service attacks in iot networks through verifiable delay functions. *CoRR*, abs/2006.01977, 2020.

[30] Mi Zhou, Xiaoming Lin, Ao Liu, and Yiying Che. An improved blockchain consensus protocol with distributed verifiable delay function. In *2021 IEEE International Conference on Electronic Technology, Communication and Information (ICETCI)*, pages 330–337, 2021.

[31] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, USA, 1996.

[32] Wei Guo, Sujuan Qin, Jun Lu, Fei Gao, Zhengping Jin, Qiaoyan Wen, and Daniele Sgandurra. Improved proofs of retrievability and replication for data availability in cloud storage. *The Computer Journal*, 63(1):1216–1230, 2020.

[33] Eliza Paul. Linux file sharing services ftp, nfs and samba. `https://www.emptrust.com/blog/benefits-of-using-digital-signatures/#:~:text=Digital%20signatures%20can%20provide%20proof,been%20tampered%20with%20during%20transit.&text=Digital%20signatures%20are%20significantly%20more,other%20forms%20of%20electronic%20signatures.`, 2017. Accessed on 30/01/2022.

[34] M.J.B. Robshaw. Security estimates for 512-bit rsa. In *Proceedings of WESCON'95*, pages 409–, 1995.

[35] Hadeal Abdulaziz Al Hamid, Sk Md Mizanur Rahman, M. Shamim Hossain, Ahmad Almogren, and Atif Alamri. A security model for preserving the privacy of medical big data in a healthcare cloud using a fog computing facility with pairing-based cryptography. *IEEE Access*, 5:22313–22328, 2017.

[36] Aniruddha Bhattacharjya, Xiaofeng Zhong, and Xing Li. A lightweight and efficient secure hybrid rsa (shrsa) messaging scheme with four-layered authentication stack. *IEEE Access*, 7:30487–30506, 2019.

[37] Raza Imam, Qazi Mohammad Areeb, Abdulrahman Alturki, and Faisal Anwer. Systematic and critical review of rsa based public key cryptographic schemes: Past and present status. *IEEE Access*, 9:155949–155976, 2021.

[38] Amazon Web Services. Aws free tier. `https://aws.amazon.com/free/?nc1=h_ls&all-free-tier.sort-by=item.additionalFields.`

SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all/, 2022. Accessed on 13/01/2022.

[39] Alireza Yazdanpanah and Mahmoud Reza Hashemi. A new compression ratio prediction algorithm for hardware implementations of lzw data compression. In *2010 15th CSI International Symposium on Computer Architecture and Digital Systems*, pages 155–156, 2010.

[40] Bertel King. Which file transfer method is best for your home network? https://www.makeuseof.com/tag/file-transfer-home-network/, 2019. Accessed on 14/01/2022.

[41] Develop Paper. Linux file sharing services ftp, nfs and samba. https://developpaper.com/linux-file-sharing-services-ftp-nfs-and-samba/, 2020. Accessed on 14/01/2022.

[42] Guohang Lu, Yi Liu, Yifei Chen, Chengwei Zhang, Yayu Gao, and Guohui Zhong. A comprehensive detection approach of wannacry: Principles, rules and experiments. In *2020 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pages 41–49, 2020.

[43] Stephen B. Wicker. The ethics of zero-day exploits—: The nsa meets the trolley car. *Commun. ACM*, 64(1):97–103, dec 2020.

[44] Ying Li, Radim Bartos, and Chunchao Liang. Are containers coupled with netem a reliable tool for performance study of network protocols? In *2019 SoutheastCon*, pages 1–7, 2019.

[45] Hemminger Stephen. Network emulation with netem. *2005 Proceedings of the 6th Australia's National Linux Conference*, pages 1–8, 04 2005.

[46] Audrius Jurgelionis, Jukka-Pekka Laulajainen, Matti Hirvonen, and Alf Inge Wang. An empirical study of netem network emulation function-

alities. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6, 2011.

[47] NetEm. Netem, the linux foundation. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`, 2021. Accessed on 11/04/2021.

[48] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing web latency: The virtue of gentle aggression. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 159–170, New York, NY, USA, 2013. Association for Computing Machinery.

[49] Michael Dahlin, Bharat Baddepudi V. Chandra, Lei Gao, and Amol Nayate. End-to-end wan service availability. *IEEE/ACM Trans. Netw.*, 11(2):300–313, April 2003.

[50] Srikanth Sundaresan, Walter de Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. Broadband internet performance: A view from the gateway. *SIGCOMM Comput. Commun. Rev.*, 41(4):134–145, August 2011.

[51] Dan Dobre, Ghassan Karame, Wenting Li, Matthias Majuntke, Neeraj Suri, and Marko Vukolić. Powerstore: Proofs of writing for efficient and robust storage. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer amp; Communications Security*, CCS '13, page 285–298, New York, NY, USA, 2013. Association for Computing Machinery.

[52] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. Understanding and mitigating packet corruption in data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 362–375, New York, NY, USA, 2017. Association for Computing Machinery.

[53] Vern Paxson. End-to-end internet packet dynamics. *SIGCOMM Comput. Commun. Rev.*, 27(4):139–152, October 1997.

[54] Red Hat. Understanding openstack. `https://www.redhat.com/en/topics/openstack`, 2021. Accessed on 05/02/2022.

[55] Zhu Kai, Liu Youyu, Lv Qi, Shi Cheng Hao, and Zhang Liping. Building a private cloud platform based on open source software openstack. In *2020 International Conference on Big Data and Social Sciences (ICBDSS)*, pages 84–87, 2020.

[56] Arsalan Saghir and Tahir Masood. Performance evaluation of openstack networking technologies. In *2019 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–6, 2019.

[57] Brandon Butler. 15 most powerful openstack companies. `https://www.networkworld.com/article/2176960/15-most-powerful-openstack-companies.html`, 2014. Accessed on 05/02/2022.

[58] Sean Michael Kerner. Openstack now powers 75 public clouds worldwide. `https://www.eweek.com/cloud/openstack-now-powers-75-public-clouds-worldwide/`, 2018. Accessed on 05/02/2022.

Appendices

Given $h_{i,j}^{(k)} = \prod_{w \in S_{i,j}^{(k)}} H(w)$ and $y_{i,j}^{(k)} = (g_{i,j}^{(k)})^{P/P_{i,j}^{(k)}} \in \mathbb{Z}_N$, so:

$$(y_{i,j}^{(k)})^{P_{i,j}^{(k)}} = ((g_{i,j}^{(k)})^{P/P_{i,j}^{(k)}})^{P_{i,j}^{(k)}} = ((h_{i,j}^{(k)})^{1/P})^P = h_{i,j}^{(k)} \mod N$$

From a challenge set $chal = (Q, R)$ and response set $resp = (\{\mu_j\}_{j \in [1,s]}, \sigma)$. We get the following

$$\begin{aligned}
\sigma &= \prod_{(i,v_i) \in Q} (\sigma_i \cdot \prod_{j=1}^{s} \prod_{k \in R} u_j^{m_{i,j}^{(k)}})^{v_i} \\
&= \prod_{(i,v_i) \in Q} \sigma_i^{v_i} \times \prod_{(i,v_i) \in Q} \prod_{j=1}^{s} \prod_{k \in R} (u_j^{m_{i,j}^{(k)}})^{v_i} \\
&= \prod_{(i,v_i) \in Q} (G(id||i) \cdot \prod_{j=1}^{s} u_j^{m_{i,j}})^{v_i d} \times \prod_{(i,v_i) \in Q} \prod_{j=1}^{s} \prod_{k \in R} (u_j^{m_{i,j} + y_{i,j}^{(k)}})^{v_i} \\
&= \prod_{(i,v_i) \in Q} (G(id||i)^{v_i} \cdot \prod_{j=1}^{s} u_j^{v_i m_{i,j}})^{d} \times \prod_{(i,v_i) \in Q} \prod_{j=1}^{s} \prod_{k \in R} u_j^{m_{i,j} v_i} \times \prod_{(i,v_i) \in Q} \prod_{j=1}^{s} \prod_{k \in R} u_j^{y_{i,j}^{(k)} v_i} \\
&= \Big(\prod_{(i,v_i) \in Q} G(id||i)^{v_i} \times \prod_{j=1}^{s} u_j^{(\sum_{(i,v_i) \in Q} v_i m_{i,j})} \Big)^d \\
&\quad \times \prod_{j=1}^{s} \prod_{k \in R} u_j^{(\sum_{(i,v_i) \in Q} v_i m_{i,j})} \times \prod_{(i,v_i) \in Q} \prod_{j=1}^{s} \prod_{k \in R} u_j^{v_i (h_{i,j}^{(k)})^{1/P_{i,j}^{(k)}}} \\
&= \Big(\prod_{(i,v_i) \in Q} G(id||i)^{v_i} \times \prod_{j=1}^{s} u_j^{\mu_j} \Big)^d \times \prod_{j=1}^{s} u_j^{|R| \mu_j} \times \prod_{(i,v_i) \in Q} \prod_{j=1}^{s} \prod_{k \in R} u_j^{v_i (h_{i,j}^{(k)})^{1/P_{i,j}^{(k)}}}
\end{aligned}$$

and so

$$\begin{aligned}
\sigma^e &= \prod_{(i,v_i) \in Q} G(id||i)^{v_i} \times \prod_{j=1}^{s} u_j^{\mu_j} \times \Big(\prod_{j=1}^{s} u_j^{|R| \mu_j} \times \prod_{(i,v_i) \in Q} \prod_{j=1}^{s} \prod_{k \in R} u_j^{v_i (h_{i,j}^{(k)})^{1/P_{i,j}^{(k)}}} \Big)^e \\
&= \prod_{(i,v_i) \in Q} G(id||i)^{v_i} \times \prod_{j=1}^{s} u_j^{\mu_j (1 + |R| e)} \times \Big(\prod_{(i,v_i) \in Q} \prod_{j=1}^{s} \prod_{k \in R} u_j^{v_i (h_{i,j}^{(k)})^{1/P_{i,j}^{(k)}}} \Big)^e \mod N
\end{aligned}$$

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

Deputy Vice-Chancellor's Office
Postgraduate Research Office

Co-Authorship Form - Masters

This form is to accompany the submission of any thesis that contains research reported in co-authored work that has been published, accepted for publication, or submitted for publication. A copy of this form should be included for each co-authored work that is included in the thesis. Completed forms should be included at the front (after the thesis abstract) of each copy of the thesis submitted for examination and library deposit.

Please indicate the chapter/section/pages of this thesis that are extracted from co-authored work and provide details of the publication or submission from the extract comes:
Chapter IV and Chapter V have been used in the paper entitled 'Efficient Publicly Verifiable Proofs of Data Replication and Retrievability Applicable for Cloud Storage' accepted for publication in the Advances in Science, Technology and Engineering Systems Journal

Please detail the nature and extent (%) of contribution by the candidate:

Certification by Co-authors:
If there is more than one co-author then a single co-author can sign on behalf of all
The undersigned certifies that:

- The above statement correctly reflects the nature and extent of the Masters candidate's contribution to this co-authored work
- In cases where the candidate was the lead author of the co-authored work he or she wrote the text

Name:	*Clementine Gritit*	Signature: CGritti		Date:	*18 February 2022*