Integrity and Privacy of Large Data

by

Jalaj Upadhyay

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctorate of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2015

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The content presented in Chapter 4 and Chapter 6 was co-authored with Maura Paterson. All the chapters of this thesis were authored under the supervision of Douglas Stinson.

Abstract

There has been considerable recent interest in “cloud storage” wherein a user asks a server to store a large file. One issue is whether the user can verify that the server is actually storing the file, and typically a challenge-response protocol is employed to convince the user that the file is indeed being stored correctly. The security of these schemes is phrased in terms of an extractor which will recover the file given any “proving algorithm” that has a sufficiently high success probability. This forms the basis of *proof-of-retrievability* (PoR) and *proof-of-data-possession* (PDP) systems. The contributions of this thesis in secure cloud storage are as below.

1. We provide a general analytical framework for various known PoR schemes that yields exact reductions that precisely quantify conditions for extraction to succeed as a function of the success probability of a proving algorithm. We apply this analysis to several archetypal schemes. In addition, we provide a new methodology for the analysis of keyed PoR schemes in an unconditionally secure setting, and use it to prove the security of a modified version of a scheme due to Shacham and Waters [72] under a slightly restricted attack model, thus providing the first example of a keyed PoR scheme with unconditional security. We also show how classical statistical techniques can be used to evaluate whether the responses of the prover on the storage are accurate enough to permit successful extraction. Finally, we prove a new lower bound on the storage and communication complexity of PoR schemes.

2. We propose a new type of scheme that we term a *proof-of-data-observability scheme*. Our definition tries to capture the stronger requirement that the server must have an actual copy of M in its memory space while it executes the challenge-response protocol. We give some examples of schemes that satisfy this new security definition. As well, we analyze the efficiency and security of the protocols we present, and we prove some necessary conditions for the existence of these kinds of protocols.

3. We study secure storage on multiple servers. Our contribution in multiple-server PoR systems is twofold. We formalize security definitions for two possible scenarios: (i) when a threshold of servers succeed with high enough probability (worst-case) and (ii) when the average of the success probability of all the servers is above a threshold (average-case). Using coding theory, we show instances of protocols that are secure both in the average-case and the worst-case scenarios.

Acknowledgements

I have been fortunate in having the benefit of interactions with numerous people to whom I feel grateful and would like to thank on this occasion. I would particularly like to thank my PhD supervisor, Prof. Douglas Stinson, for giving me support and guidance. Doug let me work freely on various topics. This thesis would not have existed without his guidance. His knowledge and way of looking at a new problem greatly influenced the way I developed as a researcher. I would also like to thank Ian Goldberg, Guang Gong, Alfred Menezes, and Daniel Panario for agreeing to be in my committee and for their useful feedbacks to improve this thesis.

I would like to thank my coauthors: Simon Blackburn, Nishanth Chandran, Vipul Goyal, Aniket Kate, Pratyay Mukherjee, Omkanth Pandey, and Maura Paterson. Each of them taught me different aspects of research. I would also like to thank Moritz Hardt for showing great confidence in me, Eric Blais for all the insightful discussions and suggestions during one of my research projects, and Shai-Ben David for teaching me machine learning and being patient with all my questions. I never had the opportunity of working on a project with them, but they worked as a great mentor to me during my last year of PhD.

I would like to thank students of the CrySP lab through all these years: Atif, Chuan, Cecylia, Colleen, Erinn, Femi, Greg, Hasan, Hooman, Jeremy, Mehrdad, Nik, Rob, Ryan, Sukhbir, Tao, and Vladimir. I will always cherish their talks in the CrySP meeting and the random discussion in the cubicles. I would like to especially thank Ryan for all the discussions on cryptography, Jeremy for his quick-witted sense of humour, and Aniket for our coffee time discussions and hosting me during my visit to Saarbucken. Aniket and Jeremy guided me a lot during the starting three years of my graduate life. I would like to thank the theory group of 2011-2013: Abel, Jamie, Raju, Ruth, Sev, and Stacey.

I would always relish the brotherly treatment rendered to me by friends of my brother. Aashish and his wife Nikita often treated me with great food, especially my favourite dish "malai-kofta." Srinath has been there for me at any hour of the day and, in retrospect, with great advices. Sachin and his sense of humour would lift my mood at any time. Niraj was always there with a funny anecdote to mark the situation.

My special list of roommates made my life during graduate studies highly entertaining. Akhilesh was always there for me with his great advice even after he moved out. He is one of guys I can call even in the middle of the night to discuss my problems. Shubham carried his friendship from the undergraduate times during his year long stay with me. I would always miss his mathematical curiosity and his way of finding new jokes to light up my mood. Rajat is one of the best roommates one could imagine. His *poha* was the

highlight of weekends, even on camping and hiking trips. He was my first squash teacher and I would like to appreciate his patience with me during the learning phase. Shitikanth has been a special friend during the last years of my stay at Waterloo. We had amazing discussions on mathematics, politics, philosophy, sociology, etc. How he could come up with simple and logically consistent arguments would always baffle me!

I made wonderful friends during my stay at Waterloo. I would always relish the time spent with Hrushikesh, Robin, and Prashant in the DC food court after our classes. We never managed to open up a company, but we did had some decent start-up ideas. I would always relish the quick witted jokes of Vinayak, the time spent with Xili and Shubham in the Williams coffee shop, the late night snacks in Prachi's office, and great conversation on literature and arts with Rose. In the later half of my graduate life, I met Anirudh, Anup, Deepak, Nikki, Meenu, Rupinder, Sandeep, Shivam, Shrinu, and Zoey. Anirudh exposed my taste-bud to a wide variety of cuisines, ranging from Iranian to African. I will always remember the time spent with Sandeep for his amazing sense of humour, Anup and Deepak for their political discussions, Meenu for her great cooking, Nikky for her sense of humour (often at my expense), Shrinu for countless hours practicing in the squash courts and discussions on differential privacy, and Zoey for waking me up in the middle of night to do karaoke. This thesis and my graduate life would have been very different without the contributions and love of Darya. Over the six years that I have known her, and even after she left Canada, Darya never let me become complacent about my research and was always there with her smile and support.

I made many friends in the squash court, who would remain my friend in and out of the court: Adam Jaffe, Adam Rauf, Andrew, Anton, Blake, Cameron, Jared, Jeff Muirhead, Jeff Porter, Luke, Michael Szestopalow, Michael Yetisir, Mohit, Nick, Pranav, Shobhit, Steven, and Tyrone. All the hours spent in the squash court always rejuvenated me and that would have been impossible without all of them. Andrew, Pranav, and Rajat helped me a lot to improve my game. I would also like to thank the varsity coaches, Clive and Vinit, for showing immense patience in improving my squash skills and giving me useful tips to recover from various injuries.

Finally, I cannot describe in words, my love towards my family. My mother has been a pillar of strength and it is her untiring support and encouragement that gave me the determination and will to pursue my endeavors. My father has been my role model and constant source of inspiration. My brother has always been with me and has supported me through all ups and downs in my life. My sister-in-law has always made me feel good with her cheerful and enthusiastic support.

To my family.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many agencies generate and many collect enormous amounts of data and store it in the form of a database either locally or remotely. Depending on how the data is handled and for what purpose the data is generated, one would like to achieve various goals. At a very high level, we can have three possible scenarios:

- The data is outsourced by an agency on a remote server just for storage purposes. In this case, the agency would like to have a certain level of guarantee that the data is stored properly.
- The data is outsourced to a remote server and then the agency wishes to perform some computations on it. In this case, the agency would like to make sure that the computation of the data is performed correctly (and that the confidentiality of the data is maintained).
- The agency just collects data of its users so that another entity called an *analyst* can perform some analysis on the stored data. In this case, the agency should itself be accountable for making sure that the users' confidential data is not leaked even if the analyst behaves maliciously. In other words, the agency is responsible to sanitized the data in such a manner so that the confidential data is not leaked under any adversarially chosen analysis.

In each of the three cases mentioned above, we have a well-defined adversary. In the first two cases, it is the remote server that perhaps acts arbitrarily, and we would like to secure our data against it. In the third setting, we consider that the analyst who performs the analysis on the data provided by the agencies acts adversarially, and our goal is to preserve privacy against any such malicious analyst. The adversary in this case has no direct access to the data, but has access to the sanitized data provided by the agency. The adversary in the first two cases has access to the whole data all the time. In other words, there is a difference in how an adversary can access the data in the various cases under consideration.

Once we have identified the type of adversary, and how and what it can access, the next important task for an agency is to construct a defence mechanism. There are trivial ways to resist attacks in the three scenarios mentioned above. For instance, to ensure that a server stores the agency's data, the agency can compute a small cryptographically secure fingerprint of its file before storing the file on the server. Then, whenever the agency wishes to access its data, it downloads the whole data and checks that the precomputed fingerprint matches the fingerprint of the downloaded data. Similarly, to ensure that the correct computation is done and the privacy of the data is maintained, the agency can encrypt the data and store it on the server. Now when it wishes to compute the function, it can request the data, decrypt it, and perform the computation locally. Likewise, one method that ensures that an individual's confidential data is not leaked when performing the aggregate analysis is to add a lot of noise to the database while performing the tasks.[1]

Unfortunately, none of these solutions is satisfactory and scalable from the application point of view. For example, in the first two cases, it might be very inefficient to download the whole data every time, especially if the data is very large.

The failure of these trivial solutions results in the following pertinent questions while handling large data: (i) how can one verify "efficiently" that files stored on possibly malicious cloud servers are stored properly, (ii) how can one verify "efficiently" that the computation performed by the remote server is done correctly on its data, and (iii) how can one answer queries made by a malicious analyst on a private database with substantial "accuracy" without leaking information about any individual in the database. The first question forms the basis of most of the recent studies in *proof-of-storage* [4, 51], the second question forms the basis of *delegation of computation* [39, 42], and the third question forms the basis of the algorithmic developments in *differential privacy* [34].

The content of this thesis covers the first question in more detail. We study various aspects of proof-of-storage. The primary aim of proof-of-storage is to assure a user that a possibly malicious server has stored its file properly.

[1] This may not be trivial as noted by Mironov [61].

1.1 Problems Studied in this Thesis

With the advent of cloud computing, more and more data is being stored by users on backup servers. Often, the user might not even have a local copy of the data. This makes verifying the integrity of the data extremely important and brings the accountability of the remote storage server into the picture. The problem is more pressing if the data is accessed only rarely. In such a scenario, the natural question is how can a client be assured that the entire outsourced data is stored properly all the time? For example, if the data is rarely accessed and the storage server loses some data inadvertently, the server might reason that there is no need to notify its clients! Alternatively, a malicious server might even choose to delete rarely accessed files for economic reasons. To assuage such concerns, a user would like to have an auditing procedure to verify that its data is stored correctly. This was addressed formally in two concurrent works resulting in the two most widely used security definitions.

1. Juels and Kaliski [51] formalized the notion of *proof-of-retrievability* (PoR) systems, in which the security guarantee captures the requirement of a file being able to be retrieved from the server through audits.

2. Atieniese *et al.* [3] formalized the notion of *proof-of-data-possession* (PDP) systems which guarantee that a server has the file during the audits.

For a scalable solution, one would like to minimize the amount of computation and the number of locations of the data a server has to access in order to respond correctly to the client. In other words, one would like to reduce the following two cost factors.

1. Communication cost. The number of bits exchanged between the server and the user during any audit protocol should be "small".

2. Storage overhead cost. The amount of extra storage overhead on both the server and the user should be small, and yet, a successful audit protocol proves beyond doubt to the user that the server has stored the file properly.

Typically, a user divides the file in the form of blocks before storing it on the server. In order for the user to be assured that the file is being stored correctly on the server, a challenge-response protocol is periodically invoked by the user, wherein the server must give a correct response to a random challenge chosen by the user. The response is typically a function of one or more file blocks. We do not assume that the user is storing the file. Therefore, in a basic version of a storage scheme that provides accountability of the server, the user must precompute and store a sufficient number of challenge-response pairs before transmitting the file to the server. The user may then erase the file and only retain the precomputed challenge-response pairs. One other possibility is that the user stores a private key and uses it to verify the correctness of the response of the server.

1.1.1 Unconditionally Secure Proof-of-Retrievability

In the recent past, many constructions of PoR-systems have been proposed. All these constructions rely heavily on standard (though unproven) underlying hardness assumptions and cryptographic primitives. For example, the original construction of Juels and Kaliski [51] uses pseudo-random functions. Likewise, the Shacham and Waters [72] scheme uses cryptographic primitives like secure pseudo-random functions and secure signature schemes. On top of that, their scheme uses hardness assumptions like RSA and the bilinear decisional Diffie-Hellman assumption. Therefore, it is natural to ask the following question:

Question 1. *Is it possible to reduce the dependence of the security of* PoR *systems on underlying hardness assumptions or the existence of secure cryptographic primitives?*

The above question motivates us to study PoR systems in the setting of unconditional security. There are several advantages of studying unconditionally secure PoR schemes. First, the schemes are easier to understand and analyze because we do not use any additional cryptographic primitives or unproven assumptions (e.g., pseudo-random functions, signatures, bilinear pairings, message-authentication codes, hitting samplers, random oracle model, etc.). This allows us to give very simple exact analyses of various schemes. Secondly, the essential role of error-correcting codes in the design and analysis of PoR schemes becomes clear: codes are not just a method of constructing PoR schemes; rather, every PoR scheme gives rise to a code in a natural way.

1.1.2 Retrievability vs Possession

The two security definitions mentioned above use the "extractor" paradigm. In this paradigm, the security guarantee requires a proof of the existence of an entity called an *extractor* that retrieves the original file from any server that passes the audits with some reasonable probability. The definitions of PoR and PDP systems both use the extractor paradigm in the same manner: if the server succeeds in the challenge-response protocol with high enough probability, then there exists an extractor that can output the file. This seems to suggest that there is little or no fundamental difference between PoR and PDP systems in the present literature. Therefore, we raise the following question.

Question 2. *What is a natural definition that captures the notion that the server has the file when it responds to the audits, and are there protocols that satisfy this definition?*

One might argue that "possessing" data should mean that the server has an exact copy of the file M in its storage at a given time. However, it would be difficult to force the server to do this, because the server can store M in some altered form, e.g., involving encryption or compression. Then the server could reconstruct M whenever it responds to a challenge by the client. However, it is conceivable that the challenge-response protocol might necessitate the reconstruction of M, in the sense that a response cannot be generated without first restoring M to its initial form. In this case, an exact copy of M would exist in the server's memory space at the time the response to the challenge is computed, and we could say that the server "possesses" the data at this point in time.

1.1.3 Multiple-server Proof-of-Retrievability

Integrity guarantees of an outsourced file on multiple servers has been studied previously. Curtmola, Khan, Burns, and Ateniese [27] studied the PDP system when multiple copies of the same file are stored at different servers and proposed a scheme secure against a restricted adversarial model. Subsequently, Bowers, Juels, and Oprea [19] studied a system where a single file is distributed and stored on different servers and modeled the security against adaptive adversaries. In some sense, these two studies can be seen as specific instances of wide variety of PoR systems that are possible when there is more than one server.

For example, Bowers, Juels, and Oprea [19] do not specify explicitly in their security definition when the extractor is supposed to succeed. To make this more precise, consider the following model: the extractor succeeds if all (or the majority of) the servers succeed with high enough probability. This model is too restrictive. This is because it may be possible that some of the servers (even a small fraction) succeed with high enough probability to compensate for the failure of the rest of the servers. Moreover, it is never clearly stated what level of interaction is allowed between the servers — are the servers allowed to interact only before, or also during the audit protocol? These basic questions make it imperative to investigate and study the formal security definition in the multi-server PoR system (MPoR). We raise and answer the following two fundamental questions that arise when a user stores its data on multiple servers.

Question 3. *What is a reasonable definition of multi-server proof-of-storage systems that reflects the natural integrity requirement of the data? Moreover, can we construct a proof-of-storage system that is secure under this definition?*

1.2 Organization of this Thesis

This thesis consists of the following chapters.

1. In Chapter 2, we give basic background of the mathematical concepts to the level required to understand this thesis.

2. In Chapter 3, we introduce various concepts relating to the proof-of-storage and some of the related works that are important in the context of this thesis.

3. In Chapter 4, we answer question 1. We provide a general analytical framework for schemes that yield exact reductions that precisely quantify the conditions for extraction to succeed as a function of the success probability of a proving algorithm. We apply this analysis to several archetypal schemes. In addition, we provide a new methodology for the analysis of keyed PoR schemes in an unconditionally secure setting, and use it to prove the security of a modified version of a scheme due to Shacham and Waters [72] under a slightly restricted attack model, thus providing the first example of a keyed PoR scheme with unconditional security. We also show how classical statistical techniques can be used to evaluate whether the responses of the server are accurate enough to permit successful extraction. Finally, we prove a new lower bound on storage and communication complexity of unconditionally secure PoR schemes.

4. In Chapter 5, we answer question 2. We propose a new type of system that we term a *proof-of-data-observability system*. Our definition tries to capture the stronger requirement that the server must have an actual copy of M in its memory space while it executes the challenge-response protocol. We give some examples of schemes that satisfy this new security definition. We also analyze the efficiency and security of the protocols we present, and prove some necessary conditions for the existence of these kinds of protocols.

5. In Chapter 6, we answer question 3. We study secure storage on multiple servers. Our contribution in MPoR systems is twofold. We formalize security definitions for two possible scenarios: (i) when a threshold of servers succeed with high enough

probability (worst-case security), and (ii) when the average of the success probabilities of all the servers is above a threshold (average-case security). Using coding theory, we show instances of protocols that are secure both in the average-case and the worst-case scenarios. We also extend the statistical techniques to help users evaluate whether the responses of the servers are good enough to allow successful extraction in the average case. We also give a generic transformation that creates a dynamic-MPoR system from any single-server PoR system.

6. In Chapter 7, we conclude this thesis by giving some open problems that we believe are important to better understand issues relating to the integrity and privacy of large data.

Chapter 2

Preliminaries

In this chapter, we give the mathematical preliminaries and notations that we use in this thesis. For the convenience of the reader, we enumerate the notations used in this thesis as an appendix at the end of this thesis.

We start with various asymptotic notations that we use in this thesis. Let n be a natural number. We use the following asymptotic notations:

- Big $O(\cdot)$ notation: $f(n) = O(g(n))$ if $\exists k > 0 \ , \exists n_0 \ , \forall n > n_0 \ , f(n) \leq k \cdot g(n)$.
- Big $\Omega(\cdot)$ notation: $f(n) = \Omega(g(n))$ if $\exists k > 0 \ , \forall n_0 \ , \exists n > n_0 \ , f(n) \geq k \cdot g(n)$.
- Big $\Theta(\cdot)$ notation: $f(n) = \Theta(g(n))$ if $\exists k_1 > 0 \ , \exists k_2 > 0 \ , \exists n_0 \ , \forall n > n_0, k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$.

2.1 Basic Algebra

We give a brief exposition of basic algebra to the level required to understand this thesis. We refer the readers to standard textbook on this topic for more details [14]. For a real number x, we denote by $|x|$ the absolute value of x. We use boldface lowercase letters to denote vectors, for example, $\mathbf{x}$, and $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to denote the entries of $\mathbf{x}$. For two vectors $\mathbf{x}$ and $\mathbf{y}$, we denote by $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_i \mathbf{x}_i \mathbf{y}_i$ the inner product of $\mathbf{x}$ and $\mathbf{y}$. We let $\mathbf{e}_1, \ldots, \mathbf{e}_n$ denote the standard basis vectors in $\mathbb{R}^n$, i.e., $\mathbf{e}_i$ has entries 0 everywhere except for the position i where the entry is 1. We denote by $\mathbf{A} \| \mathbf{b}$ the vector formed by appending the matrix $\mathbf{A}$ with the vector $\mathbf{b}$. We overload this notation to denote concatenation of two strings x and y as $x \parallel y$. We use the notation $\mathbb{I}_n$ to denote the identity matrix of order n. Where it is clear from the context, we drop the subscript.

For a $p \times N$ matrix $\mathbf{A}$, we denote by $\mathbf{A}_{ij}$ the (i, j)-th entry of $\mathbf{A}$. We denote by $\mathsf{vec}(\mathbf{A})$ the vector of length pN formed by the entries of the matrix $\mathbf{A}$, i.e., for an $p \times N$ matrix $\mathbf{A}$, the $((i-1)N + j)$-th entry of $\mathsf{vec}(\mathbf{A})$ is $\mathbf{A}_{ij}$, where $1 \leq i \leq p, 1 \leq j \leq N$. The *transpose* of a matrix $\mathbf{A}$ is a matrix $\mathbf{B}$ such that $\mathbf{B}_{ij} = \mathbf{A}_{ji}$. We use the notation $\mathbf{A}^{\mathsf{T}}$ to denote the transpose of a matrix $\mathbf{A}$. A matrix is a *cyclic matrix* if the i-th row of the matrix is formed by cyclically shifting the entries of $(i-1)$-th row one position to the right. In other words, let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be the entry of first row, then a cyclic matrix formed by $\mathbf{x}$ has the following form:

$$\begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_n \\ \mathbf{x}_n & \mathbf{x}_1 & \ldots & \mathbf{x}_{n-1} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{x}_2 & \mathbf{x}_3 & \ldots & \mathbf{x}_1 \end{pmatrix}.$$

We use the symbol $\mathbb{F}_q$ to denote a finite field of order q and $\mathbb{F}_q[x]$ to denote the ring of polynomials over $\mathbb{F}_q$. We call a polynomial random if all its coefficients are uniformly and randomly picked from the underlying field. We refer the readers to standard textbooks on this material, like Shoup [76]. For an integer n, we use the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We use the notation $\lfloor x \rfloor$ to denote the largest integer smaller than or equal to x. We use $\ln(\cdot)$ to denote the natural logarithm and $\lg(\cdot)$ to denote logarithm to the base 2.

We can represent a cyclic matrix by a polynomial. An $n \times n$ cyclic matrix over $\mathbb{F}_q$ can be represented by a degree at most $n-1$ polynomial $a(x) \in \mathbb{F}_q[x]$ such that the first row is formed by the coefficients of $a(x)$. For example, if $a(x) = 1$, then the corresponding circular matrix is the identity matrix and if $a(x) = 1 + x$, the corresponding cyclic matrix is

$$\begin{pmatrix} 1 & 1 & 0 & \ldots & 0 \\ 0 & 1 & 1 & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ 1 & 0 & 0 & \ldots & 1 \end{pmatrix}.$$

2.2 Error-correcting Codes

Error-correcting codes provide a systematic way of adding redundancy to a message so that, even if the receiver receives a corrupted message, the redundancy allows the receiver to retrieve the original message. In this section, we give a brief exposition of error-correcting codes to the level required to understand the material in this thesis. We refer the interested reader to the excellent book on this topic by MacWilliams and Sloane [59]. We start with some basic notions concerning error-correcting codes.

1. **(Encoding).** Let $k < n$. An *encoding function* defined over an alphabet Λ is a function $\mathsf{Enc} : \Lambda^k \rightarrow \Lambda^n$ that maps a *message* of k symbols to a string of n symbols over the alphabet Λ. The encoded string is called a *codeword.*

2. **(Error-correcting Code).** An *error-correcting code* is the image of the encoding function. We use the symbol C to denote the set of valid codewords (image of Enc function).

3. **(Decoding).** The *decoding function* is a function that maps a string of length n over the alphabet set Λ to a string of length k or a special symbol $\perp$ to denote the failure of decoding process.

The ratio k/n is the *rate* of the code. The length of the codeword is the *block length* of the code and the length of the message is the *message length.* In this terminology, the *rate* of a code is the ratio of the message length to the block length. The number of indices where two codewords c_1 and c_2 differ is the *Hamming distance*, or more simply, *distance* between c_1 and c_2, and is denoted by $\mathsf{dist}(c_1, c_2)$. The *minimum distance* of a code is the minimum distance between any two distinct codewords. We reserve the symbol d to denote the minimum distance of a code. The *dimension* of an error-correcting code is defined by the quantity $\log_{|\Lambda|} |C|$, which is k in our case.

One of the important goals in an error-correcting code is to decode a possibly corrupted message to the original message. Various decoding methods have been devised to correctly decode corrupted messages. One of the most popular methods of decoding is *nearest neighbour decoding.* Given a received codeword, nearest neighbour decoding finds the codeword which has minimum distance from the received codeword. This method can correct up to $\lfloor(\mathsf{d} - 1)/2\rfloor$ errors, where d is the distance of the code. It is also referred to as *minimum distance decoding* in the literature.

2.2.1 Linear Codes

An important class of error-correcting codes is called *linear codes*. We define them next.

Definition 2.1. *Let q be a prime power and let $\mathbb{F}_q$ denote the finite field with q elements. A* linear code *C, defined over the alphabet $\Lambda = \mathbb{F}_q$, of block length n and dimension k is a k-dimensional linear subspace of $\mathbb{F}_q^n$.*

We use the symbol $[n, k]_q$ to denote a linear code defined over $\mathbb{F}_q$ with block length n and message length k. A linear code has lot of nice combinatorial and structural properties. We list two of them below.

1. A linear code over $\mathbb{F}_q$ has q^k elements, where k is the dimension of the code.
2. A linear code always contains the all zero string. Hence the distance of a linear code is the minimum Hamming weight of a non-zero codeword.

We next present a well-known code family known as Reed-Solomon codes.

Reed-Solomon Codes [68]. An $[n, k]_q$ Reed-Solomon code, with $k < n \le q$, is defined as follows: Let $a_1, \ldots, a_n$ be n distinct elements from $\mathbb{F}_q$. Let $m = (m_0, \ldots, m_{k-1}) \in \mathbb{F}_q^k$ be the message. Then define a degree-k polynomial $p_m(x) = m_0 + m_1 x + \ldots + m_{k-1} x^{k-1}$. The encoding of m is the evaluation of $p_m(\cdot)$ on $a_1, \ldots, a_n$, i.e., $(p_m(a_1), \ldots, p_m(a_n))$.

Conventionally, a message is represented in form of a row vector. We will use the same convention. A succinct way to represent a linear code is the matrix that generates the codewords of the code.

Definition 2.2. *An $[n, k]_q$ linear code C, defined over the alphabet $\Lambda = \mathbb{F}_q$, can be represented by the set $\left\{ \boldsymbol{x}\mathbf{G} : \boldsymbol{x} \in \mathbb{F}_q^k \right\}$ for a $k \times n$ matrix $\mathbf{G}$ of rank k; such a matrix is called a* generator matrix *of C. Any generator matrix whose first k columns forms an identity matrix is a* generator matrix of the standard form. *Equivalently, the code can be also represented by a subspace $\{ \boldsymbol{y} : \boldsymbol{y} \in \mathbb{F}_q^n \text{ such that } \mathbf{P}\boldsymbol{y}^{\mathsf{T}} = \mathbf{0} \}$ for an $(n-k) \times n$ matrix $\mathbf{P}$ having rank $n - k$; such a matrix is a* parity check matrix *of C.*

Any generator matrix of a code can be converted to a standard form.

Example 1. The following is the generator matrix and parity check matrix of a code C that has distance 3 and can correct a single-bit error:

$$\mathbf{G} := \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{P} := \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Since this code is defined over $\mathbb{F}_2$, we perform all the arithmetic modulo 2. Therefore, to encode a message $\mathbf{x} = \begin{pmatrix} 1, & 0, & 1, & 1 \end{pmatrix}$, we compute $\mathbf{xG} \mod 2 = \begin{pmatrix} 0, & 1, & 1, & 0, & 0, & 1, & 1 \end{pmatrix}$, where $\mathbf{xG} \mod 2$ denotes coordinate-wise modulo arithmetic. Now suppose we receive a message $\mathbf{z}$ and if no error has occurred, then $\mathbf{P}\mathbf{z}^\mathsf{T} = \mathbf{0}^\mathsf{T}$. For example, if we receive $\mathbf{z} = \begin{pmatrix} 0, & 1, & 1, & 0, & 0, & 1, & 1 \end{pmatrix}$ then $\mathbf{P}\mathbf{z}^\mathsf{T} \mod 2 = \begin{pmatrix} 0, & 0, & 0 \end{pmatrix}^\mathsf{T}$. Now suppose an error has occurred at some position, say the fifth position, and $\mathbf{z} = \begin{pmatrix} 0, & 1, & 1, & 0, & 1, & 1, & 1 \end{pmatrix}$ is received, then $\mathbf{P}\mathbf{z}^\mathsf{T} = \begin{pmatrix} 1, & 0, & 1 \end{pmatrix}^\mathsf{T}$. This corresponds to the fifth column of $\mathbf{P}$. From this, we conclude that the error has occurred at the fifth position. Mathematically, we have $\mathbf{P}\mathbf{z}^\mathsf{T} = \mathbf{P}\mathbf{e}_5^\mathsf{T}$ in this case.

A parity check matrix gives a nice characterization of the distance of the corresponding code.

Fact 2.1. *The distance of a code is the minimal number of linearly dependent columns of its parity check matrix.*

An important concept in linear codes is the concept of *dual code*. For a linear code C, its dual code, denoted by $C^\perp$, is defined as the set of codewords

$$C^\perp = \{\mathbf{x} : \langle \mathbf{x}, \mathbf{c} \rangle = 0 \text{ for all } \mathbf{c} \in C\}.$$

The distance of a dual code is the *dual distance* of C. We reserve the symbol $\mathbf{d}^\perp$ to denote the dual distance. The parity check matrix for a code C is the generator matrix for its dual code $C^\perp$. A *self-dual code* is a code for which $C = C^\perp$. An $[n,k]_q$ code is a *maximal distance separable* code if it satisfies the property $\mathsf{d} = n-k+1$. The dual code of a maximal distance separable linear code has a distance $\mathsf{d}^\perp = k+1$.

In general, the decoding algorithm requires the whole codeword to output the message. However, when we only need some specific entry of the message, one can do better. There are some known constructions of error-correcting codes in which the decoding algorithm does not need to read the whole codeword. In fact, it can decode a particular entry of the message by reading a very small number of the entries of the codeword. Such codes are *locally decodable codes* [89].

Definition 2.3. *Let* loc *be a fixed constant. An* $[n,k]_2$ *code defined over the alphabet* $\Lambda = \{0,1\}$ *is an* $[n,k,\mathsf{loc}]$-locally decodable code *if there exists a decoding algorithm that, on input an index* $1 \le i \le k$*, needs at most* loc *positions of a codeword to output the bit at the* i*-th position of the encoded message with high probability.*

2.2.2 Low-density Parity-check Codes

Low-density parity-check (LDPC) codes were introduced by Gallager [38] in his doctoral thesis. They are defined by $(N-k) \times N$ parity-check matrices. Let φ and ζ be small constants. An LDPC code is defined as the null space of a parity check matrix $\mathbf{P}$ with the following structural properties: (1) each column of $\mathbf{P}$ consists of at most φ non-zero entries; (2) each row of $\mathbf{P}$ consists of at most ζ non-zero entries; and (3) both φ and ζ are "small" compared to N and $N-k$, respectively. The constant φ is the *row sparsity* and the constant ζ is the *column sparsity* of the LDPC code corresponding to $\mathbf{P}$. For example, in Example 1, the column sparsity for C is $\zeta = 3$.

There are many constructions of LDPC codes, including algebraic constructions [29, 38], probabilistic constructions [13], and constructions based on finite geometry [55]. In what follows, we explore one probabilistic construction and one based on finite geometry.

Bennatan and Burshtein [13] analyzed three constructions of LDPC codes. The first and the second constructions are based on the original construction of Gallager [38], while the third construction is defined over $\mathbb{F}_q$ for a prime number q. In this thesis, we will be mainly interested in the construction over $\mathbb{F}_q$, for which they proved the following theorem.

Theorem 2.2. [13] *Let* $\zeta > 3$ *and* φ *be arbitrary constant natural numbers. Let* $R = 1-\zeta/\varphi$ *and let* q *be a prime. Then there exists a probabilistic construction of an* $(N-k)\times N$ LDPC *matrix with row sparsity* ζ *and column sparsity* φ *such that there exists* $c_1 > 0$ *and a universal constant* c_2 *such that*

$$\mathsf{Pr}[\mathsf{d} \le c_1 N] = c_2 N^{1-\zeta/2}.$$

In the above theorem, the explicit bound on c_1 is as follows:

$$\inf_{c \ge 3} \left\{ e^{-12+6\ln(\zeta(q-1)/(1-R))/c} \right\}.$$

For an explicit combinatorial construction, we have to resort to the finite geometry based codes. We need some notation before we give these results. Let $\mathbb{EG}(u, 2^w)$ denote the u-dimensional vector space of all 2^{uw} u-tuples over $\mathbb{F}_{2^w}$. A line in $\mathbb{EG}(u, 2^w)$ is a one-dimensional subspace that consists of 2^w points. $\mathbb{EG}(u, 2^w)$ is called an u-dimensional Euclidean geometry over $\mathbb{F}_{2^w}$. The following is a well-known result regarding codes based on Euclidean geometry.

Theorem 2.3. *Let u and w be positive integers, where $w \geq 2$. Then there is an explicit construction of a parity-check matrix for an* LDPC *code with 2^{uw} columns and $(2^{(u-1)w})(2^{uw} - 1)/(2^w - 1)$ rows and*

$$\mathsf{d} = \frac{2^{uw} - 1}{2^w - 1} + 1, \quad \mathsf{d}^{\perp} = (2^w + 2)2^{w(u-2)}, \quad \varphi = \frac{2^{uw} - 1}{2^w - 1}, \quad \textit{and } \zeta = 2^w.$$

Proof. Let the parity check matrix be the incidence matrix whose columns are points in $\mathbb{EG}(u, 2^w)$ and rows are the lines in $\mathbb{EG}(u, 2^w)$. Then the number of columns is 2^{uw} and the number of lines that intersects at any point in $\mathbb{EG}(u, 2^w)$ are $\frac{2^{us}-1}{2^w-1}$. Therefore, the number of rows are

$$\frac{2^{(u-1)w}(2^{uw} - 1)}{2^w - 1}.$$

Finally, any line passes through 2^w points, which is the row sparsity of the incidence matrix. The dual distance of this code follows from Calkin *et al.* [21]. Since the distance of a code is the smallest number of linear dependent columns of the parity-check matrix and two columns have only one 1 in common, it follows that the minimum number of linearly dependent columns is $\varphi + 1$. The result follows. □

We remark that this construction can be extended to any finite field $\mathbb{F}_{q^w}$ for any prime $q \geq 2$.

2.2.3 Linear Codes with Low-density Generator Matrices

The construction of Kou, Lin, and Fossorier [55] based on Euclidean geometry yields a parity-check matrix which has bounded row and column sparsity. However, the corresponding generator matrix in the standard form may not be sparse. In this thesis, we would need a code whose generator matrix has bounded row and column sparsity when represented in the standard form.[1] Such a construction was given by Johnson and Weller [50] based on *difference families*. We first define a difference family and then state their result.

[1] The condition that the columns and rows of the standard generator matrix are sparse may not always be possible by considering the dual code from any low density parity-check matrices formed in Section 2.2.2.

Definition 2.4. *Let G be an additive group of order v. Then L r-subsets of G, $B_i = \{b_{i,1}, \ldots, b_{i,r}\}$ for $1 \leq i \leq \mathsf{L}$ form a (v, r, λ)*-difference family *if every nonzero element of G occurs exactly λ times among $b_{i,x} - b_{i,y}$ for $1 \leq i \leq \mathsf{L}$ and $1 \leq x, y \leq r$.*

Example 2. The sets $\{\{0, 1, 7, 11\}, \{0, 2, 3, 14\}, \{0, 4, 6, 9\}\}$ form a $(19, 4, 2)$-difference family over $\mathbb{Z}_{19}$.

Johnson and Weller [50] showed the following theorem.

Theorem 2.4. *Let $D_1, \ldots, D_{\mathsf{L}}$ be a (v, r, λ)-difference family. Then there is a parity-check matrix with row sparsity $(\mathsf{L} - 1)r + 1$ and column sparsity r.*

Construction 1. The construction of Johnson and Weller [50] has the following form

$$\mathbf{P} := \begin{pmatrix} \mathbf{C}_1 & \mathbf{C}_2 & \ldots & \mathbf{C}_{\mathsf{L}} \end{pmatrix},$$

where $\mathbf{C}_1, \ldots, \mathbf{C}_{\mathsf{L}}$ are $v \times v$ cyclic matrices. If one of these matrices is invertible, say $\mathbf{C}_1$, then the corresponding generator matrix is

$$\mathbf{G} := \begin{pmatrix} & (\mathbf{C}_1^{-1}\mathbf{C}_2)^{\mathsf{T}} \\ & (\mathbf{C}_1^{-1}\mathbf{C}_3)^{\mathsf{T}} \\ \mathbb{I}_{v(\mathsf{L}-1)} & \vdots \\ & (\mathbf{C}_1^{-1}\mathbf{C}_{\mathsf{L}-1})^{\mathsf{T}} \\ & (\mathbf{C}_1^{-1}\mathbf{C}_{\mathsf{L}})^{\mathsf{T}} \end{pmatrix}$$

Recall that a cyclic matrix can be completely characterized by the polynomial

$$a(x) = a_0 + a_1 x + \ldots + a_{v-1} x^{v-1},$$

where the coefficients are the entries in the first row.

Let $a_1(x), \ldots, a_{\mathsf{L}}(x)$ be the L polynomials corresponding to $\mathbf{C}_1, \ldots, \mathbf{C}_{\mathsf{L}}$. For $\mathbf{C}_1$ to be an invertible cyclic matrix, we need to ensure that the greatest common divisor of $a_1(x)$ and $x^v - 1$ is 1. This can be easily achieved by many choices of polynomials. An example is $a_1(x) = 1$, which corresponds to the identity matrix.

Example 3. Consider the construction of a code with $v = 5, \mathsf{L} = 2$. This code is a rate 1/2 code. Let $a_1(x) = 1 + x$ and $a_2(x) = 1 + x^2 + x^4$. Note that $a_2(x)^{-1} \equiv (x^2 + x^3 + x^4) \bmod (x^5 - 1)$. The generator matrix consists of a 5×5 identity matrix and a 5×5 cyclic matrix defined by the polynomial $1 + x^3 \equiv (a_2^{-1}(x)a_1(x)) \mod (x^5 - 1)$. In other words, we have the following generator and parity check matrices:

$$\mathbf{G} := \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \qquad \mathbf{P} := \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

The generator matrix has a row sparsity 3 and the parity check matrix has a row sparsity 5. The column sparsity of the generator matrix is 2 while that of the parity check matrix is 3.

If we use $a_1(x) = 1$, which corresponds to $\mathbf{C}_1$ being an identity matrix, we get the result stated in Theorem 2.4. More details follow.

Let $D_2, \dots, D_{\mathsf{L}}$ be $\mathsf{L} - 1$ sets of a $(v, r, 1)$-difference family such that

$$D_i := \{d_{i,1}, \dots, d_{i,r}\}, \qquad \text{for } 2 \le i \le \mathsf{L}.$$

The polynomial $a_i(x)$ for the cyclic matrix $\mathbf{C}_i$ for $2 \le i \le \mathsf{L}$ is defined as follows:

$$a_i(x) := x^{d_{i,1}} + x^{d_{i,2}} + \ldots + x^{d_{i,r}}.$$

To ensure that $\mathbf{C}_1$ is invertible, we use $a_1(x) = 1$.

Using the observation that the distance of a code is the minimal number of linearly dependent columns of its parity check matrix and that $\mathbf{G}\mathbf{P}^{\mathsf{T}}$ is an all-zero matrix, we get the following result.

Lemma 2.5. *Let $\mathbf{G}$ be a generator matrix corresponding to the parity-check matrix defined in Theorem 2.4. Then $\mathbf{G}$ is a generator matrix for a code with message length $v(\mathsf{L} - 1)$ and block length $v\mathsf{L}$, such that $\mathsf{d} = \mathsf{L}r$ and $\mathsf{d}^{\perp} = r + 2$ with $\mathsf{L} \ge 2$.*

It is well known that a $(v, r, 1)$-difference family exists for all $v \equiv 1 \mod 6$ and $r = 3$ [25].

2.3 Information Theory

Information theory is a branch of mathematics that allows one to quantify information [26]. In this section, we give a brief overview of the central concepts in information theory.

1. **(Shannon Entropy).** A key measure of information is the *Shannon entropy*, which measures the amount of information in a message over a specified alphabet. The Shannon entropy $\mathbf{H}$ of a discrete random variable X taking values in Λ is defined as follows:
$$\mathbf{H}(X) := -\sum_{x \in \Lambda} p(x) \log_2(p(x)),$$
where $p(x)$ is the probability of symbol x. A fundamental property of entropy is that $0 \leq \mathbf{H}(X) \leq \log_2 |\Lambda|$, where the second equality holds if and only if all the symbols are equally likely.

2. **(Joint Entropy).** The *joint entropy* of two discrete random variables X and Y is defined as follows:
$$\mathbf{H}(X;Y) := -\sum_{x,y} p(x,y) \log_2(p(x,y)).$$
A fundamental property of joint entropy is that $\mathbf{H}(X;Y) \leq \mathbf{H}(X) + \mathbf{H}(Y)$ with equality if and only if the two random variables X and Y are independent.

3. **(Conditional Entropy).** The *conditional entropy* measures the uncertainty of random variable X given the occurrence of random variable Y.
$$\mathbf{H}(X|Y) := -\sum_{y} p(y) \sum_{x} p(x|y) \log_2(p(x|y)) = \mathbf{H}(X;Y) - \mathbf{H}(Y).$$
From the definition it is easy to see that $\mathbf{H}(X|Y) \leq \mathbf{H}(X)$.

4. **(Mutual Information).** *Mutual information* between two random variables measures the amount of information one can obtain about one variable by observing the other variable. By definition it is symmetric. Given two random variables X and Y, the mutual information of X relative to Y is
$$\begin{aligned} I(X;Y) &= \mathbf{H}(X) - \mathbf{H}(X|Y) \\ &= \mathbf{H}(X) + \mathbf{H}(Y) - \mathbf{H}(X;Y) \\ &= \mathbf{H}(Y) - \mathbf{H}(Y|X) \\ &= I(Y;X). \end{aligned}$$

2.4 Secret Sharing Schemes and their Variants

In this thesis, we use various primitives related to secret sharing schemes. A *secret sharing scheme* allows a trusted dealer to share a secret between n players so that certain subsets of players can reconstruct the secret from the shares they hold [15, 73]. We start the discussion of secret sharing schemes by first defining the most basic form of secret sharing scheme, the *threshold secret sharing scheme* [31].

Definition 2.5. *Let τ and n be positive integers such that $\tau \leq n$. A (τ, n)-*threshold secret sharing scheme *is a pair of algorithms:* $(\mathsf{ShareGen}, \mathsf{Reconstruct})$ *such that, on input a secret* S*,* $\mathsf{ShareGen}(\mathsf{S})$ *generates n shares, one for each of the n players, such that the following two properties hold: (i) Reconstruction: any subset of τ or more players can pool together their shares and use* $\mathsf{Reconstruct}$ *to compute the secret s from the shares that they collectively hold, and (ii) Secrecy: no subset of fewer than τ players can determine any information about the secret* S*.*

One can also define a threshold secret sharing scheme in the terms of entropy. Let us denote the secret by the random variable S and let S_A be the probability distribution on the vectors of shares of a subset $A \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$.

Definition 2.6. *Let Υ consist of all subsets of $\{1, \ldots, n\}$ of size at least τ. A (τ, n)-*threshold secret sharing scheme *satisfies the following properties: (i) for every set $A \in \Upsilon$, $\mathbf{H}(\mathsf{S}|\mathsf{S}_A) = 0$, and (ii) for every $B \notin \Upsilon$, $\mathbf{H}(\mathsf{S}|\mathsf{S}_B) = \mathbf{H}(\mathsf{S})$.*

A classic example of a threshold secret sharing scheme is Shamir's secret sharing scheme [73]. We first recall the Lagrange interpolation method. Let q be a prime or a prime power. Let $x_1, \ldots, x_\tau$ be τ distinct elements in the finite field $\mathbb{F}_q$. Let $a_1, \ldots, a_\tau$ be arbitrary elements in $\mathbb{F}_q$. Then there is a unique polynomial $f(x) \in \mathbb{F}_q[x]$ of degree at most $\tau - 1$ such that $f(x_i) = a_i$ for all $1 \leq i \leq \tau$, namely:

$$f(x) := \sum_{i=1}^{\tau} \left(\prod_{1 \leq j \leq \tau, j \neq i} \frac{x - x_j}{x_i - x_j} a_i \right).$$

This formula for the unique polynomial $f(x)$ is called *Lagrange's interpolation formula.*

We can now describe Shamir's secret sharing scheme. The dealer picks a finite field $\mathbb{F}_q$ such that $q > n$. In the first stage, the dealer picks a random polynomial $f(x) \in \mathbb{F}_q[x]$ of degree at most $\tau - 1$ such that its constant term is the secret s that needs to be shared. It then sends the share $f(i)$ to player i. During the reconstruction phase, a subset of τ players S come together and they compute

$$f(0) = \sum_{i \in S} \left(\prod_{j \in S, j \neq i} \frac{j}{j - i} f(i) \right),$$

using the Lagrange's interpolation formula by setting $x = 0$.

Example 4. Let $q = 31, \tau = 3, n = 8$. Let the secret be $s = 7$. Then the dealer picks a random polynomial $f(x) = 7 + 19x + 21x^2 \in \mathbb{Z}_{31}[x]$. The dealer then computes the shares set $(16, 5, 5, 16, 7, 9, 22, 15)$ and gives the i-th share to $\mathcal{P}_i$. Now suppose $\mathcal{P}_1, \mathcal{P}_5$, and $\mathcal{P}_7$ come together, then they can compute the secret as follows:

$$f(0) \equiv \left(\frac{16 \cdot 5 \cdot 7}{(1-5)(1-7)} + \frac{7 \cdot 1 \cdot 7}{(5-1)(5-7)} + \frac{22 \cdot 1 \cdot 5}{(7-1)(7-5)} \right) \mod 31 = 7.$$

A threshold secret sharing scheme is a special case of a general secret sharing scheme. In order to give the general definition of secret sharing scheme, we need to first define *access structures.*

Definition 2.7. *Let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be a set of n players. An access structure Υ is a set of subsets of players that satisfies the following two conditions: (i) if $A \in \Upsilon$ and $A \subseteq B \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, then $B \in \Upsilon$, and (ii) if $A \in \Upsilon$, then $|A| > 0$. The first property is called the* monotone *property.*

Example 5. Let us consider four parties with indices $\{1, 2, 3, 4\}$. The set of subsets $\Upsilon := \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 2, 3, 4\}\}$ is a monotone access structure. On the other hand, the set of subsets $\Upsilon' := \{\{1, 2\}, \{3, 4\}\}$ is non-monotonic, because $\{1, 3, 4\}$ is not included.

We can now give a more formal definition of secret sharing scheme using the concept of access structures [12]. Let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be a set of n players. We assume that there is a probability distribution whose support is the set of possible secrets.

Definition 2.8. *Let n be a positive integer and let Υ be an access structure on the set $\{\mathcal{P}_1, \cdots, \mathcal{P}_n\}$. A (Υ, n)-*secret sharing scheme *is a pair of algorithms denoted by* $(\mathsf{ShareGen}, \mathsf{Reconstruct})$*, such that, on input a secret* S, $\mathsf{ShareGen}(\mathsf{S})$ *generates n shares, one for each of the n players, such that the following two properties hold: (i) Reconstruction: if the set of players specified by any set in Υ pool together their shares, they can use* $\mathsf{Reconstruct}$ *to compute the secret s from the shares that they collectively hold, and (ii) Secrecy: no subset not in Υ can determine any information about the secret* S.

Just as in the case of threshold secret sharing, we can also give an entropy-based definition of a secret sharing scheme. Let us denote the secret by the random variable S and let S_A be the probability distribution on the vectors of shares of a subset $A \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$.

Definition 2.9. *Let Υ be an access structure as above. We say a* secret scheme *realizes an access structure Υ, if the following properties hold: (i) for every set $A \in \Upsilon$, $\mathbf{H}(\mathsf{S}|\mathsf{S}_A) = 0$, and (ii) for every $B \notin \Upsilon$, $\mathbf{H}(\mathsf{S}|\mathsf{S}_B) = \mathbf{H}(\mathsf{S})$.*

It is well known that the size of each players' share in a secret sharing scheme must be at least the size of the secret. If the secret that is to be shared is large, then this constraint can be very restrictive. Schemes for which we can get a certain form of trade-off between share size and security are known as *ramp schemes* [16].

Definition 2.10. (Ramp Scheme). *Let $\tau_1, \tau_2,$ and n be positive integers such that $\tau_1 < \tau_2 \leq n$. A* (τ_1, τ_2, n)-ramp scheme *is a pair of algorithms:* $(\mathsf{ShareGen}, \mathsf{Reconstruct})$ *such that, on input a secret* S*,* $\mathsf{ShareGen}(\mathsf{S})$ *generates n shares, one for each of the n players, such that the following two properties hold: (i) Reconstruction: any subset of τ_2 or more players can pool together their shares and use* $\mathsf{Reconstruct}$ *to compute the secret s from the shares that they collectively hold, and (ii) Secrecy: no subset of τ_1 or fewer players can determine any information about the secret* S*.*

Example 6. Suppose the dealer wishes to set up a $(2, 4, n)$-ramp scheme with the secret (a_0, a_1). The dealer picks a finite field $\mathbb{F}$ with size greater than n such that $a_0, a_1 \in \mathbb{F}$. The dealer picks random elements a_2, a_3 independently from the field $\mathbb{F}$ and construct the following polynomial of degree 3 over the finite field $\mathbb{F}$: $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$. The share for any player $\mathcal{P}_i$ is generated by computing $s_i = f(i)$. It is easy to see that if two or fewer players come together, they do not learn any information about the secret, and if at least four players come together, they can use Lagrange's interpolation formula to compute the function f as well as the secret. However, if three players pool together their shares, then they can learn some partial information about one of the other player's share. For concreteness, let $\mathbb{F} = \mathbb{F}_{17}$. Then $5a_1 \equiv 7s_3 + 9s_6 + s_{15} \mod 17$; therefore, players $\mathcal{P}_3, \mathcal{P}_6,$ and $\mathcal{P}_{15}$ can compute the value of a_1.

We can also give an entropy-based definition of a ramp scheme.

Definition 2.11. *Let $\mathcal{P} := \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ be a set of participating players. We say a scheme is a (τ_1, τ_2, n)-ramp scheme, denoted by* S*, if the following properties hold: (i) if $A \subseteq \mathcal{P}$ and $|A| \geq \tau_2$, then $\mathbf{H}(\mathsf{S}|\mathsf{S}_A) = 0$, and (ii) if $A \subseteq \mathcal{P}$ and $|A| \leq \tau_1$, then $\mathbf{H}(\mathsf{S}|\mathsf{S}_A) = \mathbf{H}(\mathsf{S})$.*

The traditional definition of a *threshold scheme* is a ramp scheme with $\tau_1 = \tau_2 - 1$. Linear codes have been used to construct secret sharing and ramp schemes for over thirty years since the work of McEliece and Sarwate [60]. We will consider a construction from an arbitrary code in this thesis. The following relation between an arbitrary code (linear or non-linear) and a ramp scheme was shown by Paterson and Stinson [65].

Theorem 2.6. *Let $\mathcal{C}$ be a code of length N, distance d and dual distance $\mathsf{d}^\perp$. Let $1 \leq \mathsf{s} < \mathsf{d}^\perp - 2$. Then there is a $(\tau_1, \tau_2, N - \mathsf{s})$ ramp scheme, where $\tau_1 = \mathsf{d}^\perp - \mathsf{s} - 1$ and $\tau_2 = N - \mathsf{d} + 1$.*

Here s is the *rate* of the ramp scheme. If $\mathbf{G}$ is a generator matrix of a code C of dimension k, then $|C| = q^k \geq q^{\mathsf{d}^\perp - 1}$. In other words, $k \geq \mathsf{d}^\perp - 1$.

Construction 2. The construction of a ramp scheme from a code is as follows. Let s and ρ be positive integers and let $(m_1, \ldots, m_{\mathsf{s}}) \in \mathbb{F}^{\mathsf{s}}$ be the message. Let C be a code of length $n = \rho + s$ defined over a finite field $\mathbb{F}$. We also require that the first s entries of a codewords is the message to be encoded, i.e., the corresponding generator matrix is in the standard form. Select a random codeword $(\mathbf{c}_1 = m_1, \ldots, \mathbf{c}_{\mathsf{s}} = m_{\mathsf{s}}, \mathbf{c}_{\mathsf{s}+1}, \ldots, \mathbf{c}_{\rho+\mathsf{s}}) \in C$, and define the shares as $(\mathbf{c}_{\mathsf{s}+1}, \ldots, \mathbf{c}_{\rho+\mathsf{s}})$.

Example 7. One can use Reed-Solomon code to construct a ramp scheme. Let q be a prime and $1 \leq \mathsf{s} < t \leq n < q$. We know for a prime q, that there is an $[N, k, N - \tau + 1]_q$ Reed-Solomon code with $\mathsf{d}^\perp = \tau + 1$. This implies a $(\tau - \mathsf{s}, \tau, N)$-ramp scheme over $\mathbb{F}_q$.

2.4.1 Linear Secret Sharing Scheme

A ramp scheme constructed using a linear code as in Example 7 has a linearity property, i.e., for any two secrets S and S' and respective share vectors $(\mathsf{S}_1, \mathsf{S}_2, \ldots, \mathsf{S}_n)$ and $(\mathsf{S}'_1, \mathsf{S}'_2, \ldots, \mathsf{S}'_n)$, the vectors $(\mathsf{S}_1 + \mathsf{S}'_1, \mathsf{S}_2 + \mathsf{S}'_2, \ldots, \mathsf{S}_n + \mathsf{S}'_n)$ and $(\lambda\mathsf{S}_1, \lambda\mathsf{S}_2, \ldots, \lambda\mathsf{S}_n)$ are valid share vectors for the secrets $\mathsf{S} + \mathsf{S}'$ and $\lambda\mathsf{S}$ respectively. A ramp scheme with this linearity property is called a (τ_1, τ_2, n)-*linear secret sharing scheme*. In other words, linear secret-sharing schemes allow one to share a secret among n people so that if the secret is changed using some linear function, then one can compute the share corresponding to the new secret by applying the same function on all the shares.

If a ramp scheme is constructed from a linear code as above, then for every two secrets, the corresponding vectors of shares differ, in the Hamming sense, by at least d, the distance of the underlying code.

When $\mathsf{j} = 1$, we have the following instantiation.

Example 8. Let C be a linear code of length $N+1$ defined over a finite field $\mathbb{F}$. Let $\mathsf{S} \in \mathbb{F}$ be the secret. Select a codeword $(\mathbf{c}_0 = \mathsf{s}, \mathbf{c}_1, \ldots, \mathbf{c}_n) \in C$, and define the shares as $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$. The result is a $(\mathsf{d}^\perp - 2, N - \mathsf{d} + 2, N)$-linear secret sharing scheme, where $\mathsf{d}^\perp$ is the dual distance of C.

We use the notation $\mathsf{LSSS}(C)$ to denote a linear secret sharing scheme based on a code C. When all the codewords in C are given, the encoding function, $\mathsf{Enc}(\cdot)$ is very simple and is as defined in Construction 2.

However, in most cases we have access only to the generator matrix of the underlying code. The encoding function, Enc, in this case is defined as follows. Let the message to shared be $m = (m_1, \ldots, m_{\mathsf{s}}) \in \mathbb{F}_q^{\mathsf{s}}$ using a ramp scheme based on a $[k, \rho + \mathsf{s}]$ linear code C with generator matrix $\mathbf{G}$. We can assume that the generator matrix is in standard form without any loss of generality because a generator matrix can be easily converted to its standard form as per Definition 2.2. The encoding function now picks random entries $\mathbf{x}_j \in \mathbb{F}_q$ for $j \geq \mathsf{s} + 1$ and constructs $\mathbf{x} = (m_1, \ldots, m_{\mathsf{s}}, \mathbf{x}_{\mathsf{s}+1}, \ldots, \mathbf{x}_k)$. We then compute $\mathbf{c} = \mathbf{x}\mathbf{G}$. The shares are $(\mathbf{c}_{\mathsf{s}+1}, \ldots, \mathbf{c}_{\rho+\mathsf{s}})$.

The following example illustrates the encoding function when $k = \mathsf{s}$.

Example 9. Let

$$\mathbf{G} := \begin{pmatrix} 1 & 0 & 0 & 9 & 2 & 11 & 7 \\ 0 & 1 & 0 & 4 & 11 & 12 & 11 \\ 0 & 0 & 1 & 8 & 9 & 4 & 11 \end{pmatrix}$$

be a generator matrix of a $[3, 7]_{13}$-linear code defined over $\mathbb{F}_{13}$. Let $\mathbf{x} = \begin{pmatrix} 10, & 3, & 5 \end{pmatrix} \in (\mathbb{F}_{13})^3$ be the secret. In other words, $\mathsf{j} = k$. Then $\mathbf{x}\mathbf{G} = \begin{pmatrix} 10, & 3, & 5, & 12, & 7, & 10, & 2 \end{pmatrix}$ and 12 is the share of $\mathcal{P}_1$, 7 is the share of $\mathcal{P}_2$, 10 is the share of $\mathcal{P}_3$, and 2 is the share of $\mathcal{P}_4$.

Chapter 3

Key Concepts in Proof-of-Storage

This chapter introduces two types of proof-of-storage systems: *proof-of-retrievability* (PoR) systems and *proof-of-data-possession* (PDP) systems. Both of these concepts try to capture two natural requirements of secure cloud storage: PoR captures the requirement that the file can be "retrieved" from a server provided the server responds correctly to a large fraction of challenge-response pairs, while PDP captures the notion that the storage server has the file if it responds correctly to a large fraction of challenge-response pairs. Both of these concepts are related to a well-established notion in cryptography called *proof-of-knowledge* [9]. We start the discussion of secure cloud storage by first presenting the essence of the definition of a proof-of-knowledge system.

Proof-of-knowledge. Proof-of-knowledge (PoK) systems are defined for binary relations. A binary relation is a subset $R \subseteq \{0,1\}^* \times \{0,1\}^*$. We say that R is *polynomially bounded* if there exists a polynomial function $\mathrm{poly}(\cdot)$ such that $|y|_b \leq \mathrm{poly}(|x|_b)$, where $|\cdot|_b$ denotes the bit-length of its argument, for all $(x, y) \in R$. We say that R is an NP *relation* if it is polynomially bounded and there exists a polynomial time algorithm for deciding membership of a given pair (x, y) in R. If R is a binary relation, we let $R(x) := \{y : (x, y) \in R\}$ and $L_R := \{x : \exists y \text{ such that } (x, y) \in R\}$. L_R is called the *language* corresponding to the binary relation. The first coordinate in R is called the *instance* and any $y \in R(x)$ is called a *witness* for the instance x. For a given instance, there can be more than one witness.

A *proof-of-knowledge* for a relation R is a two-party interactive protocol between a Prover and a Verifier with the following components. The two parties share a common instance x of the relation R and the Prover claims to know a witness corresponding to the instance x. A protocol is a proof-of-knowledge when the following is satisfied: if the

Verifier accepts the proof provided by the Prover that the instance belongs to R, then the Prover indeed knows a witness $y \in R(x)$. This has been defined formally by a paradigm known as the *extractor* paradigm. To understand the need of this paradigm, we recall the motivating discussion by Goldreich [40, Section 4.7.1.1]:

> "What does it mean to say that a MACHINE knows something? Any standard dictionary suggests several meanings for the verb to know, and most meanings are phrased with reference to awareness, a notion that is certainly inapplicable in our context. We must look for a behavioristic interpretation of the verb to know. Indeed, it is reasonable to link knowledge with ability to do something, be it (at the least) the ability to write down whatever one knows. Hence, we shall say that a machine knows a string a if it can output the string a. But this seems total nonsense: A machine has a well-defined output – either the output equals a or it does not. So what can be meant by saying that a machine can do something? Loosely speaking, it means that the machine can be easily modified so that it will do whatever is claimed. More precisely, it means that there exists a machine that, using the original machine as a black box, outputs whatever is claimed."

The above discussion leads to the definition of a *knowledge extractor*. A knowledge extractor is a machine that is given access to a program, which specifies the behaviour of the Prover in response to a particular challenge it receives. This program is also called a *proving algorithm*. In the end, the extractor should output what the prover asserts as its knowledge at the start of the two-party protocol. Formally, Bellare and Goldreich [9] formulated the security definition of proof-of-knowledge systems as follows:

> "Let η be the probability with which the Verifier accepts, on instance x, when interacting with a Prover. A system is a *proof-of-knowledge system* if there exists a probabilistic polynomial time machine Extractor and a polynomial poly$(\cdot)$ such that Extractor runs in expected polynomial time[1] and outputs a $y \in R(x)$ with

[1] The definition of expected polynomial time [11, 18, 48] is not as straightforward as defining a polynomial time algorithm. This can be seen through the following example. Suppose, we consider an algorithm that runs in polynomial time on average as expected polynomial time, then this definition is not closed under reductions. For example, let y be a fixed string and a function $f(x) = 2^{|x|}$ if $x = y$ and $f(x) = |x|^2$ otherwise. If n is the bit length of x, then $\mathbb{E}[f(x)] < n^2 + 1$, but $\mathbb{E}[f(x)^2] > 2^n$. To remedy this, we use a linear on average function ℓ. An algorithm runs in an *expected polynomial time* if its running time is bounded by a function $f(\cdot)$ with the following property: there exists a linear-on-the-average function ℓ and a polynomial $p(\cdot)$ such that $f(n) \leq p(\ell(n))$ for all sufficiently large n. The reason why we have an average linear function $\ell(\cdot)$ is to express expected polynomial time as a function that has running time bounded by a polynomial which has an average linear growth rate.

probability at least $\eta/\operatorname{poly}(|x|_b)$."

The above definition was developed through a series of works [20, 36, 37, 81]. We discuss the definition provided by Fiat and Shamir [37] and Fiat, Fiege, and Shamir [36] and its relation to the proof-of-retrievability and proof-of-data-possession.

We give a simple protocol for proof of knowledge of *graph isomorphism*. In the example below, we we use the symbol $\mathcal{G} := (V, E)$ to denote a graph defined over a vertex set V with edge set E. The prover claims that two graphs $\mathcal{G}_1 := (V_1, E_1)$ and $\mathcal{G}_2 := (V_2, E_2)$ are *isomorphic*, i.e., one can be transformed to another by relabelling the vertices. The instance in this case consists of the two graphs $(\mathcal{G}_1, \mathcal{G}_2)$ and a witness is a bijective function $\phi : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(\phi(u), \phi(v)) \in E_2$.

Example 10. Let $\mathcal{G}_1 := (V_1, E_1)$ and $\mathcal{G}_2 := (V_2, E_2)$ be two graphs on N vertices and suppose Prover claims that the two graphs are isomorphic. The proof-of-knowledge of graph isomorphism is as follows. Prover selects a random isomorphic copy of $\mathcal{G}_2$ and sends it to the Verifier, i.e., it picks a random permutation π on V_2 and constructs a graph with vertex set V_2 and edge set $E_3 := \{(\pi(u), \pi(v)) : (u, v) \in E_2\}$. Let this graph be $\mathcal{G}$. The Verifier selects $\sigma \in \{1, 2\}$ and ask the Prover to show an isomorphism between $\mathcal{G}_\sigma$ and $\mathcal{G}$. If $\sigma = 1$, the Prover sends the composition of the two functions $\pi \circ \phi$, else it sends π to the Verifier. The Verifier can now check the isomorphism between $\mathcal{G}_\sigma$ and $\mathcal{G}$ using the function it receives from the Prover. The knowledge extractor of this protocol is described in detail in [40, Section 4.7.6.2].

3.1 Proof-of-storage System

A proof-of-storage system has two entities: a server and a client. Proof-of-storage systems are intended to guarantee that a server honestly stores the file of a client. In order for the client to be assured that the file is being stored correctly by the server, it periodically invokes a challenge-response protocol, wherein the server must give a correct response to a random challenge chosen by the client. This response will typically be a function of one or more file blocks. We do not assume that the client is storing the file. Therefore, in the basic version of any scheme that provides a proof-of-storage, the client must precompute and store a sufficient number of challenge-response pairs, before transmitting the file to the server. After this is done, the client erases the file but retains the precomputed challenge-response pairs. In a more sophisticated system, the client need not precompute the challenge-response pairs, but can generate challenges during the time of audits. For such schemes to work, typically, the client has a small secret key which is used to verify the correctness of the responses made by the server. It is also possible that the client could retain a copy of the file, in which case the responses do not need to be precomputed. This might be done if the server is just being used to store a backup copy of the file.

Depending on who can initiate an audit and verify the correctness of the audits, one can have two types of proof-of-storage systems: *publicly verifiable* proof-of-storage systems and *privately verifiable* proof-of-storage systems. In a privately verifiable proof-of-storage system, a client either stores a secret key or has some number of precomputed challenge-response pairs. In a publicly verifiable proof-of-storage system, the client has a public key corresponding to the private key of the server. The server generates a response that uses the private key in a non-trivial manner and the client (or any other entity who wishes to verify the integrity of the file) can use the public key to verify whether the responses are generated correctly.

In a typical scheme, the file is "encoded" using an error-correcting code such as a Reed-Solomon code before it is sent to the prover. The code provides redundancy, enabling correction of erasures or corrupted file blocks.

Remark 1. We use the term Prover to identify any server that stores the file of a client. We use the term Verifier for any entity that verifies whether the file of a client is stored properly or not by the server. We also assume that a file is composed of message blocks of an appropriate fixed length. If the file consists of a single block, we simply call that block the file. □

The security definition of proof-of-storage systems follows closely the definition of proof-of-knowledge systems; however, there are some key differences, on which we elaborate next in more detail.

Difference Between Proof-of-knowledge and Proof-of-storage Systems

The definition of Bellare and Goldreich [9] for proof-of-knowledge is very broad, yet there are subtle differences between the setting of proof-of-storage systems and proof-of-knowledge systems, which means we cannot use the knowledge extractor of Bellare and Goldreich [9] to define a proof-of-storage system. We list some of these differences below.

1. No natural binary relation: A proof-of-knowledge system is defined with respect to a binary relation. The knowledge extractor is required to output a $y \in R(x)$. In the setting of proof-of-storage, there is no such instance-witness pair. In other words, it is unclear if there is the natural pair whose first coordinate is the common knowledge

of the Prover and Verifier and the second coordinate is known only to the Prover. One can consider the file that the Verifier stores on the Prover as the second coordinate in the binary relation, but there is nothing specific which can act as the first coordinate.

2. Absence of common strings: Since there is no natural binary relation in the case of proof-of-storage systems, there is no set of defined instance-witness pairs. Moreover, it is not typical that the Prover and the Verifier share any randomness. More specifically, the Prover has the encoded file, while the Verifier may store some fingerprints, a secret key, or a set of of challenge-response pairs for the verification of audits. On the other hand, in the setting of proof-of-knowledge schemes, the Prover and Verifier share a common string that represents the instance to be verified and some additional randomness.

In the rest of this chapter, we discuss how to handle these two issues so that we can use the extractor paradigm to define proof-of-storage systems.

3.1.1 Proof-of-retrievability

A proof-of-retrievability (PoR) system incorporates a challenge-response protocol in which a Verifier can check that a file is being stored correctly, along with an *extractor* that will actually reconstruct the file, given a proving algorithm that responds correctly to a sufficiently high percentage of challenges.

PoR systems can be classified based on whether or not the verifier has a secret key which it uses to verify the response of the prover during a challenge-response protocol. A PoR scheme is a *keyless scheme* if the verifier does not have a secret key; otherwise, it is called a *keyed scheme*.

3.1.2 Security Definition

As we mentioned earlier, our first goal is to quantify the security afforded by the Verifier by engaging in the challenge-response protocol. Here, "security" means that the file can be correctly retrieved. The goal is that a Prover who can respond correctly to a large proportion of challenges somehow "knows" (or can compute) the contents of the file (i.e., all the message blocks). This is formalised through the notion of an *extractor*, which takes as input a description of a "proving algorithm" $\mathcal{P}$ for a certain unspecified file, and then outputs the file.

In order to generate the proving algorithm, we provide the Prover with access to the verification algorithm. We want to capture the case that a proving algorithm, which is correct with a probability sufficiently close to 1, will allow the extractor to determine the correct file. The probability that the proving algorithm $\mathcal{P}$ gives a correct response for a randomly chosen challenge is denoted by $\mathsf{succ}(\mathcal{P})$. We assume that $\mathcal{P}$ always gives some response, so it follows that $\mathcal{P}$ will give an incorrect response with probability $1 - \mathsf{succ}(\mathcal{P})$.

To summarise, we list the components in a PoR scheme we use. Note that we are employing standard models developed in the literature by various works [19, 51, 72].

- The Verifier has a message $m \in (\mathbb{F}_q)^k$ which he redundantly encodes (typically using an error-correcting code) as $M \in (\mathbb{F}_q)^n$.
- M is given to the Prover. In the case of a keyed scheme, the Prover may also be supplied with an additional tag, S.
- The Verifier retains appropriate information to allow him to verify responses. This may or may not include a key K.
- Some number of challenges and responses are carried out by the Prover and Verifier. In each round, the Verifier chooses a challenge and gives it to the Prover, and the Prover computes a response which is returned to the Verifier. The Verifier then verifies if the response is correct.
- The computations of the Prover are described by a proving algorithm $\mathcal{P}$.
- The success probability of $\mathcal{P}$ is the probability that it gives a correct response when the challenge is chosen uniformly at random.

The Extractor is given $\mathcal{P}$ and (in the case of a keyed scheme) K, and outputs an unencoded message $\hat{m}$. Extraction succeeds if $\widehat{m} = m$.

Definition 3.1. *A* PoR *system is an* (η, ν)-PoR *if there exists an* Extractor *such that the* Extractor *succeeds with probability at least* ν *whenever the success probability of* $\mathcal{P}$ *is at least* η.

In this thesis, we only consider schemes where $\nu = 1$, that is, the case when the extraction is always successful when the success probability of $\mathcal{P}$ is at least η.

There are many ways in which we can classify PoR systems. We list some of them below.

1. Based on the run time of Extractor: Dodis, Vadhan, and Wichs [33] classified PoR systems into two categories based on the run time of the extractor.

   Knowledge Soundness: We say a PoR system has *knowledge soundness* if the Extractor runs in expected polynomial time in the size of the message and security parameter.

   Information Soundness: We say a PoR system has *information soundness* if there is no restriction on the run-time of the Extractor.

2. Access to the proving algorithm: There are two ways an extractor can access the proving algorithm: either in a *black-box* manner or in a *non-black-box* manner. In a black-box setting, we allow the Extractor to access the proving algorithm by providing it with input and allowing it to observe the output — it cannot see how the proving algorithm operates. In the non-black box setting, the Extractor can see the internal working of the proving algorithm. In this thesis, we only consider extractors that access the proving algorithm in a *black-box* manner.

3. Number of challenges: If a PoR system has an *a priori* defined polynomial bound on the number of audits that can be made, then it is called a *bounded* PoR *system*; otherwise it is called an *unbounded* PoR *system.* Note that the term *unbounded* does not mean that there is no restriction on the number of audits, it only means that this bound is not specified *a priori.*

3.1.3 Proof-of-data-possession

Another type of proof-of-storage systems studied widely in the literature are *proof-of-data-possession* (PDP) systems. This notion was formalized concurrently to PoR systems by Ateniese *et al.* [3]. A PDP system permits the possibility that not all of the message blocks can be reconstructed. A *proof of data possession* (PDP) system has the same setup as that of proof-of-retrievability defined in the last section. In what follows, we use the definition given by Ateniese *et al.* [3].

Security Definition

The security definition of a PDP system is defined in terms of the knowledge extractor of Bellare and Goldreich [9]. To give a formal security definition, Ateniese *et al.* [3] used a game-based definition. It is a game (denoted by $\mathsf{Game}_{\mathsf{PDP}}$) between an adaptive Adversary

- **Setup phase:** If we are in the keyed setting, the Challenger runs the key generation algorithm to output (sk, pk) and sends pk to the Adversary and keeps sk secret.
- **Learning phase:** If we are in the keyed setting, then the Adversary makes queries to generate tags on the messages of its choice. It uses these responses to generate a proving algorithm $\mathcal{P}$ which it uses to reply to challenges in the challenge-response phase. It also stores all the tags and the message blocks in an arbitrary manner.
- **Challenge-response phase:** The Challenger generates a challenge and sends it to the Adversary. The Adversary responds to the challenge using $\mathcal{P}$.

An Adversary is said to win $\mathsf{Game}_{\mathsf{PDP}}$ if the Challenger outputs 1 after the verification of the response of the adversary.

Figure 3.1: The Security Game of PDP System ($\mathsf{Game}_{\mathsf{PDP}}$).

and a Challenger. We enumerate the key steps of the game in Figure 3.1. The guarantee that the prover actually possesses the data is captured through the following formalization given by Atienese *et al.* [3].

Definition 3.2. *A* PDP *system guarantees data possession if Adversary wins* $\mathsf{Game}_{\mathsf{PDP}}$ *with probability negligibly close to the probability that* Challenger *can extract those file blocks by using a knowledge extractor of Bellare and Goldreich [9].*

Our main criticism of Definition 3.2 is the use of the knowledge extractor in the security definition.

3.2 A Brief Look at the Security Definitions

The two definitions of proof-of-storage systems have a lot of similarities with two earlier proposed definitions of proof-of-knowledge systems by Fiege and Shamir [37] and Fiat, Fiege, and Shamir [36]. Bellare and Goldreich [9] compared in detail the two definitions for proof-of-knowledge systems. Their discussion raises the question whether the definitions for PoR and PDP are the same or different from any practical point of view. We explore this matter in more detail in Chapter 5. Here, we just mention the similarities of the security definitions of PoR and PDP systems with two older definitions of PoK systems.

In what follows, let κ be a security parameter, let R be a binary relation, let x be an instance of the relation R, let z be the auxiliary input to the Prover, and let r be the randomness shared by Prover and Verifier. Let $(\mathsf{Prover}(z) \leftrightarrow \mathsf{Verifier})(x, r, \kappa)$ denote the random variable which is 1 if and only if the Verifier accepts after its interaction with the Prover with an instance x, and 0, otherwise. We use the notation A^B to denote that an entity (or algorithm) A has access to an algorithm B in a black-box manner.

3.2.1 PoR and the PoK Systems of Fiat, Fiege, and Shamir

We next show how the definition of PoR as stated in the literature closely resembles the definition of PoK as given by Fiat, Fiege, and Shamir [36] We first state the definition of PoK for a relation R as given by Fiat, Fiege, and Shamir [36].

> For every constant $a > 0$, there exists a probabilistic polynomial time extractor Extractor so that for all constant $b > 0$, all Prover, and all sufficiently large x, r, κ, if
>
> $$\mathsf{Pr}[(\mathsf{Prover}(z) \leftrightarrow \mathsf{Verifier})(x, r, \kappa) = 1] > \kappa^{-a},$$
>
> then
>
> $$\mathsf{Pr}[\mathsf{Extractor}^{\mathcal{P}}(x, r, k) \in R(x)] > 1 - \kappa^{-b},$$
>
> where $\mathcal{P}$ is the algorithm used by the Prover to respond to the challenges.

The constants a and b signify that the probabilities are non-negligible. If we compare this definition with Definition 3.1 by considering $\eta = \kappa^{-a}$ and $\nu = 1 - \kappa^{-b}$, then it is easy to observe that Definition 3.1 is a modified form of Fiat, Fiege, and Shamir's definition of PoK system, taking into account the differences between proof-of-storage systems and PoK systems mentioned earlier in Section 3.1.

3.2.2 PDP and the PoK Systems of Fiege and Shamir

Now we consider the similarity between the formal definition of PDP and the definition of PoK given by Fiege and Shamir [37]. Fiege and Shamir defined a PoK for a relation R in the following manner.

There exists a probabilistic expected polynomial time extractor, $\mathsf{Extractor}$ such that for all Prover and all sufficiently large n, r, κ,

$$| \mathsf{Pr}[(\mathsf{Prover}(z) \leftrightarrow \mathsf{Verifier})(x, r, \kappa) = 1] - \mathsf{Pr}[\mathsf{Extractor}^{\mathcal{P}}(x, r, z) \in R(x)] | \leq \mathsf{negl}(\kappa),$$

where $\mathsf{negl}(\kappa)$ is any function that grows more slowly than any inverse polynomial function in κ and $\mathcal{P}$ is the algorithm used by the Prover to respond to the challenges.

If we compare this with Definition 3.2, it is quite clear that the definition of PDP is closely related to the definition of PoK given by Fiege and Shamir [37]. The only difference is that the Verifier in Definition 3.2 does not have access to an instance x and there is no shared randomness. These two points are exactly what we mentioned earlier in Section 3.1. In other words, the present security definition of PDP systems can be seen as a modified version of an earlier definition of PoK systems.

The differences that we mentioned in Section 3.1 raise a foundational issue in the definition of PDP systems, which in turn leads to many unresolved questions. Definition 3.2 refers to a knowledge extractor of Bellare and Goldreich [9] to give the security definition. Basically, they require that the knowledge extractor extracts the file from the proving algorithm. However, the knowledge extractor of Bellare and Goldreich [9] is defined for a relation and it is required to output a witness corresponding to the instance. To output the witness, the knowledge extractor can rewind the proving algorithm to a certain point, but it is allowed only to use the transcript of the messages exchanged between the Prover and Verifier. In the context of PDP systems, this can be troublesome for many reasons. We cite the two most important reasons here. The first issue of contention is that there is no binary relation, so the input of the knowledge extractor is not clear. The second issue is that in most practical scenarios, the Verifier would first encrypt and then outsource its data to the server's memory. In this case, no extractor can extract the file with access only to the transcript of the communication between the Prover and the Verifier unless it can break the underlying encryption scheme. For these reasons, we prefer to base our discussion on the notion of PoR.

3.3 Previous Related Work

There have been two concurrent lines of work on proof-of-storage — one aiming to construct PoR systems and one to construct PDP systems. The first construction of a PoR system was proposed by Juels and Kaliski [51]. They encrypt the file and randomly embed a set of random-valued check blocks called *sentinels*. The aim of encryption is to render the sentinels indistinguishable from other file blocks. In order to protect deletion of a small fraction of the data by a server, they also encode their encrypted data embedded with sentinels using an error-correcting code. The Verifier challenges the Prover by specifying the positions of a collection of sentinels and the Prover is required to return the associated sentinel values. The idea behind the security of their system is that if the Prover has modified or deleted a substantial portion of the file, then with high probability it will also have suppressed a number of sentinels.

Ateniese *et al.* [3] introduced the idea of using *homomorphic authenticators* to give a PDP system that has significantly less communication cost. Intuitively, a homomorphic authenticator allows tags corresponding to a message to be generated in such a manner so that the Prover can compute a valid authentication tag on the (correct) result of a certain class of functions over the message blocks. Their schemes sample the server's storage, accessing a random subset of blocks. In doing so, the scheme provides a probabilistic guarantee of possession. This scheme was improved in a follow-up work by Ateniese *et al.* [6]. Shacham and Waters [72] later showed that the scheme of Ateniese *et al.* [3] can be transformed into a PoR scheme by constructing an extractor that extracts the file from the responses of the Prover on the audits. We use their scheme extensively; therefore, we give a formal description of their scheme in Figure 3.2. Shacham and Waters [72] also instantiated their scheme to give both publicly verifiable and privately verifiable PoR systems.

Bowers, Juels, and Oprea [19] extended the idea of Juels and Kaliski [51] and used error-correcting codes. The main difference in their construction is that they use the idea of an "outer" and an "inner" code (in the same vein as concatenated codes), to get a good balance between the extra storage overhead and computational overhead in responding to the audits. Dodis, Vadhan, and Wichs [33] provided the first example of an unconditionally secure PoR scheme, also constructed from an error-correcting code, with extraction performed through *list decoding* in conjunction with the use of an *almost-universal hash function*. They also give different constructions depending on the computational capabilities of the server.

There have been some other works that provide proof-of-storage. Ateniese *et al.* [7] used *homomorphic identification schemes* to give efficient proof-of-storage systems. Wang *et al.* [87] gave the first privacy preserving public auditable proof-of-storage systems. We refer readers to the survey by Kamara and Lauter [52] regarding the architecture of proof-of-storage systems.

Dynamic PoR. One of the desirable features of cloud-based storage is to allow updating of the outsourced file at any given time. A PDP system that allows efficient updates is called a *dynamic* PDP *system.* Likewise, a PoR system that allows efficient updates is called a *dynamic* PoR *system.*

Erway *et al.* [35] gave the first dynamic-PDP systems. Wang *et al.* [88] further improved the communication cost of this scheme. However, despite these efforts, construction of a PoR scheme that allows dynamic update to the stored file eluded researchers until recently. Cash *et al.* [23] resolved this question using oblivious RAM [23]. This was further improved by Shi *et al.* [75] who used Merkle trees to give a more efficient dynamic-PoR. Subsequently, Chandran *et al.* [24] defined and constructed a locally decodable and locally updatable code, and showed how to construct a dynamic-PoR system using such codes.

Distributed Proof-of-storage. Proof-of-storage systems have been also studied in the setting where there is more than one server or more than one client. The first such setting was studied by Curtmola *et al.* [27]. They studied a multiple-replica PDP system, which is the natural generalization of a single-server PDP system to t servers. Their scheme works as follows. Let $\{f_k\}_{k \in \mathcal{K}}$ be a family of pseudo-random functions. The Verifier picks two pseudo-random functions, say $f_{K_1}(\cdot)$ and $f_{K_2}(\cdot)$, and an encryption scheme. The Verifier first generates an encryption of the file and divides it into n blocks $m := (m_1, \ldots, m_n)$. It then processes m to generate t replicas as follows: $b_{i,j} := m_i + f_{K_1}(i \parallel j)$. The Verifier then generates tags for all the t replicas in the same manner as Shacham and Waters using $f_{K_2}(\cdot)$. The audit is done exactly in the same manner as described in the description of Shacham-Waters' scheme.

Bowers *et al.* [19] introduced a distributed system, which they called HAIL. Their system allows a set of provers to prove the integrity of a file stored by a client. The idea in HAIL is to exploit the cross-prover redundancy. They considered an active and mobile adversary that can corrupt the whole set of provers.

Recently, Ateniese *et al.* [5] considered the problem from the client side, where n clients store their respective files on a single prover in a manner such that the verification of the integrity of a single client's file simultaneously gives the integrity guarantee of the files of all the participating clients. They called such a system an *entangled cloud storage.*

Input: The Verifier gets the message m. Let $\mathcal{M}$ be the message space and $\mathcal{M}^*$ be the space of encoded messages.

Initialization Stage: The Verifier performs the following steps for storing the message:

1. Let $\{f_k\}_{k \in \mathcal{K}}$ be a family of pseudo-random functions. [a] Pick a random key $K \in \mathcal{K}$ and a secret $\mathsf{a} \in \mathbb{F}_q$.
2. Pick a function $e : \mathcal{M} \rightarrow \mathcal{M}^*$. Compute $e(m) = M$ and break it into the form of n blocks, so $M = (M[1], \dots, M[n])$. Compute $S[i] = f_K(i) + \mathsf{a}M[i]$ for $1 \leq i \leq n$ and give the set of n-tuples $\{M[i], S[i]\}_{1 \leq i \leq n}$ to the Prover.

Audit Phase: During the audit phase, the Verifier interacts with the Prover as described in the following protocol:

1. The Verifier picks a challenge $V \in \mathbb{F}_q^n$ having Hamming weight ℓ. It sends the challenge V to the prover.
2. The Prover treats M and S as vectors in $\mathbb{F}_q^n$ and computes $\mathsf{R}_1 = \langle M, V \rangle$ and $\mathsf{R}_2 = \langle S, V \rangle$. The response of the Prover is the pair $(\mathsf{R}_1, \mathsf{R}_2)$.
3. The Verifier checks whether $\mathsf{R}_2 = \mathsf{a}\mathsf{R}_1 + \langle V, B \rangle$, where B is the vector $(f_K(1), \dots, f_K(n))$.

[a] A *pseudo-random function family* is a family of functions with the following property: no efficient algorithm can distinguish (with significant advantage) between a function chosen randomly from the pseudo-random function family and a function whose outputs are fixed completely at random.

Figure 3.2: Shacham-Waters Scheme for a Single-prover PoR System [72].

Chapter 4

Unconditionally Secure Proof-of-retrievability Systems

The focus of this chapter is the general construction of extractors for PoR schemes in the setting of *unconditional security.* This is a departure from most previous schemes that were secure only in setting of the computational security. This raises the natural question: why should one consider the setting of unconditional security when there are many known efficient constructions of computationally secure PoR schemes? There are several reasons why we believe it is important to analyze unconditionally secure PoR schemes. For example, our main results (Theorem 4.1) show close connections between error-correcting codes and unconditionally secure PoR systems. In addition to this, it is easier to understand and analyze unconditionally secure PoR schemes because we do not rely on an underlying hardness assumptions or cryptographic and complexity-theoretic primitives (for example, pseudo-random functions, signatures, bilinear pairings, message authentication codes, hitting samplers, random oracle model, etc.).

4.1 Our Contribution

In this chapter, we compute the precise conditions under which extraction is possible when the servers have unbounded computational power. We show that extraction in this setting can be interpreted naturally as nearest-neighbour decoding in a certain code (which we call a "response code"). This gives us a new methodology to analyze the exact security of PoR schemes. In the past, error-correcting codes have been used in specific constructions of

PoR schemes. Concisely, our result shows that error-correcting codes are not just a method of constructing PoR schemes, but that any PoR scheme secure against a computationally unbounded adversary can be seen as a code in a natural way.

We quantify the security of a PoR scheme by specifying a value η and proving that the extraction process will *always* be able to extract the file, given a proving algorithm $\mathcal{P}$ with success probability $\mathsf{succ}(\mathcal{P}) > \eta$. This is in contrast with the earlier works where the extractor is allowed to fail with some specified probability close to 0. We use this main result to analyze various PoR schemes in both the keyed as well as keyless setting.

The content of this chapter is based on the following paper [66]:

> Maura B. Paterson, Douglas R. Stinson, and Jalaj Upadhyay. A coding theory foundation for the analysis of general unconditionally secure proof-of-retrievability schemes for cloud storage. Journal of Mathematical Cryptology, 7(3):183–216, 2013.

Organization of the Chapter. This chapter is organized as follows.

- In Section 4.2, we derive bounds on the success probability of a prover in an arbitrary keyless challenge-response protocol that allows successful extraction. We use this as the basis to analyze various instantiations of keyless PoR schemes in Section 4.3. The bounds that we compute for these schemes are exact, but somewhat complicated to state. In Section 4.5.1, we give an estimate that is far easier to compute.

- The situation for keyed schemes is somewhat more complicated. For instance, consider the scheme of Shacham and Waters [72] modified appropriately to the setting of unconditional security. For this scheme, we note in Section 4.4 that if the adversary is given the verification oracle, then we cannot have unconditional security. We also show that, even in the absence of a verification oracle, one cannot prove anything non-trivial for any proving algorithm. On the other hand, we can prove non-trivial results if we analyze the success probability of a proving algorithm in the average case, over the set of keys that are consistent with the information given to the prover. This allows us to construct (and analyze) the first unconditionally secure unbounded-use keyed PoR scheme.

- The bounds in Section 4.2 and 4.4 state that successful extraction can be accomplished whenever $\mathsf{succ}(\mathcal{P})$ exceeds some pre-specified threshold. But this raises the practical question as to how the user is able to estimate $\mathsf{succ}(\mathcal{P})$. This is because in practice, a user interacts with the server only through the challenge-response protocol. In Section 4.5, we use classical statistical techniques that can help us determine whether or not the responses of the prover are accurate enough to permit successful extraction.

- Our construction for an unbounded-use unconditionally secure PoR scheme requires a Verifier to store a lot of secret information. We show in Section 4.6 that a significant additional storage requirement cannot be avoided in the setting of unconditional security by proving a new lower bound on storage and communication requirements of PoR schemes. This improves the information-theoretic lower bound for memory checkers and authenticators proven in Naor and Rothblum [62].

4.1.1 Comparison with Dodis, Vadhan, and Wichs

There are many previous works that use concepts in error-correcting codes to provide efficient PoR schemes. In the context of the construction of efficient PoR schemes, Dodis, Vadhan, and Wichs [33] remarked,

> "there is a clear relation between our problem and the erasure/error decoding of error-correcting codes."

This chapter is in some sense a general exploration of these relations, extending the work of Dodis, Vadhan, and Wichs [33]. However, there are several key differences between our approach and that of Dodis, Vadhan, and Wichs [33]:

- In the setting of unconditional security, Dodis, Vadhan, and Wichs [33] only provide bounded-use schemes. Our scheme is the first unbounded-use scheme in this setting.

- Dodis, Vadhan, and Wichs [33] mainly use a (Reed-Solomon) code to construct a specific PoR scheme. In contrast, we study the connections between an arbitrary PoR scheme and the distance of the (related) code that describes the behaviour of the scheme on the possible queries that can be made to the scheme. Stated another way, our approach is to derive a code from a PoR scheme, and then to prove security properties of the PoR scheme as a consequence of properties of this code.

- Dodis, Vadhan, and Wichs [33] use various tools and algorithms to construct their PoR schemes, including Reed-Solomon codes, list decoding, almost-universal hash families, and hitting samplers based on expander graphs. We just use an error-correcting code in our analyses.

- We base our analyses on nearest-neighbour decoding (rather than list decoding, which was used in Dodis, Vadhan, and Wichs [33]).

- We work under a stronger requirement than Dodis, Vadhan, and Wichs [33]. More concretely, we require extraction to succeed with probability equal to 1, whereas in [33], extraction succeeds with probability close to 1, depending in part on properties of a certain class of hash functions used in the protocol.

- The "PoR codes" in Dodis, Vadhan, and Wichs [33] are actually protocols that consist of challenges and responses involving a prespecified number of file blocks; we allow challenges in which the responses depend on an arbitrary number of file blocks.

- We work in the non-asymptotic setting, giving exact and concrete bounds, whereas the analyses in [33] are asymptotic.

4.2 Unkeyed PoR Schemes: the General Result

The simplest form of a PoR scheme is the one proposed by Juels and Kaliski [51]. In this scheme, the Verifier stores a random position of the encoded file and challenges the server on it during the audits. This scheme is very simple, but it only allows a bounded number of audits and requires storage on the client proportional to the number of audits it would like to make during the entire duration when the file is stored on the server. In order to make PoR schemes more efficient, we need a more general form of challenge-response protocol. For example, we might consider a response that is computed as a function of one or more file blocks.

To study the PoR system in the general case, we need to consider arbitrary challenges and their corresponding responses. We aim to be as general as possible in the setup. Let Γ denote the *challenge space* from which a challenge is picked. Let $\mathcal{M}^*$ denote the space of all encoded messages. Let $\varrho : \mathcal{M}^* \times \Gamma \to \Delta$ be the *response function*, which computes the response $\textsc{Resp} = \varrho(M, \textsc{Chal})$ given the encoded file M and the challenge $\textsc{Chal}$. We call the codomain of ϱ the *response space*.

We assume that the Adversary outputs a deterministic proving algorithm $\mathcal{P}$. The requirement of a deterministic proving algorithm is without any loss of generality. This follows from the observation that any probabilistic proving algorithm can be replaced by a deterministic algorithm relative to which the success of the extractor defined in our main result would not decrease (this would be done by hardwiring the random bits used in the probabilistic proving algorithm). This would increase the computation time by a factor exponential in the number of random bits used by the probabilistic algorithm.

The *success probability* of $\mathcal{P}$ is defined to be

$$\mathsf{succ}(\mathcal{P}) = \mathsf{Pr}_{\textsc{Chal} \in \Gamma}[\mathcal{P}(c) = \varrho(M, c)],$$

where $M = e(m)$.

For an encoded file $M \in \mathcal{M}^*$, we define the *response vector*

$$\vec{r}_M = (\varrho(M, \textsc{Chal}) : \textsc{Chal} \in \Gamma) \tag{4.1}$$

as the set of all possible responses for a given encoded file.

We define the *response code* (or more simply, the *code*) of the scheme to be the set

$$\mathcal{R}^* = \{\vec{r}_M : M \in \mathcal{M}^*\}$$

of all the response vectors corresponding to all possible files.

The Generalized Scheme is presented in Figure 4.1. Observe that $\mathcal{R}^* \subseteq \Delta^\gamma$, where $\gamma = |\Gamma|$. We will assume that the mapping $M \mapsto \vec{r}_M$ is an injection, and therefore the Hamming distance of $\mathcal{R}^*$ is greater than 0 (in fact, we make this assumption for all the schemes we consider in this chapter).

The following theorem relates the success probability of the extractor to the Hamming distance of the response code and the size of the challenge.

Theorem 4.1. *Let d^* be the Hamming distance of the response code $\mathcal{R}^*$ and γ be the size of the challenge space of the Generalized Scheme. Suppose that $\mathcal{P}$ is a proving algorithm for the Generalized Scheme such that $\mathsf{succ}(\mathcal{P}) > 1 - \mathsf{d}^*/(2\gamma)$. Then there is an Extractor that will always output $\widehat{m} = m$.*

Proof. In order to prove the above theorem, we construct an Extractor that will take as input a proving algorithm $\mathcal{P}$ for some unknown file m and output the same file $m \in \mathcal{M}$. We next describe the Extractor.

1. On input $\mathcal{P}$, compute the vector $R' = (r'_{\textsc{Chal}} : \textsc{Chal} \in \Gamma)$, where $r'_{\textsc{Chal}} = \mathcal{P}(\textsc{Chal})$ for all $\textsc{Chal} \in \Gamma$ (i.e., for every $\textsc{Chal}$, $r'_{\textsc{Chal}}$ is the response computed by $\mathcal{P}$ when it is given the challenge $\textsc{Chal}$).

2. Find $\widehat{M} \in \mathcal{M}^*$ so that $\mathsf{dist}(R', \vec{r}_{\widehat{M}})$ is minimised.

Initialization

Given a *file* $m \in \mathcal{M}$, encode[a] m as $e(m) = M \in \mathcal{M}^*$. The Verifier gives M to the Prover. The Verifier also generates a random challenge $\textsc{Chal} \in \Gamma$ and stores $\textsc{Chal}$ and $\varrho(M, c)$.

Challenge and Response: During an audit, the verifier and the prover interacts as follows.

- The Verifier gives the challenge $\textsc{Chal}$ to the Prover.
- The Prover responds with $\textsc{Resp} = \varrho(M, \textsc{Chal})$.
- The Verifier checks that the value $\textsc{Resp}$ returned by the Prover matches the stored value $\varrho(M, \textsc{Chal})$.

[a] We assume that the encoding function $e(\cdot)$ is deterministic and invertible (its purpose is solely to add redundancy).

Figure 4.1: Generalized Scheme

3. Output $\widehat{m} = e^{-1}(\widehat{M})$.

We now prove that the output of the Extractor is actually m. Let R' be the γ-tuple of responses computed by $\mathcal{P}$ and denote $\delta = \mathsf{dist}(\vec{r}_M, R')$, where $M = e(m)$. Denote $\eta = \mathsf{succ}(\mathcal{P})$. Then it is easy to see that $\eta = 1 - \delta/\gamma$. We want to prove that $\widehat{M} = M$. We have that $\vec{r}_{\widehat{M}}$ is a codeword in $\mathcal{R}^*$ closest to R'. Since M is a codeword such that $\mathsf{dist}(\vec{r}_M, R') = \delta$, it must be the case that $\mathsf{dist}(\vec{r}_{\widehat{M}}, R') \leq \delta$. By the triangle inequality, we get

$$\mathsf{dist}(\vec{r}_M, \vec{r}_{\widehat{M}}) \leq \mathsf{dist}(\vec{r}_M, R') + \mathsf{dist}(\vec{r}_{\widehat{M}}, R') \leq \delta + \delta = 2\delta.$$

However,

$$2\delta = 2\gamma(1 - \eta) < 2\gamma\left(\frac{\mathsf{d}^*}{2\gamma}\right) = \mathsf{d}^*.$$

Since $\vec{r}_M$ and $\vec{r}_{\widehat{M}}$ are codewords within distance d^*, it follows that $M = \widehat{M}$ and the Extractor outputs $m = e^{-1}(M)$, as desired. □

We take a moment to take a closer look at the above theorem. First note that Theorem 4.1 gives us information soundness and not knowledge soundness, i.e., we do not aim to construct an Extractor that runs in expected-polynomial time. Secondly, the above theorem relates the success of the extraction process with the *relative distance* of the response code $\mathcal{R}^*$, which equals d^*/γ. The relative distance of a code is one of the important properties of a code, and is an important parameter when the efficiency of a coding scheme is considered.

To better understand the statement of the theorem, we resort to the example protocol mentioned at the start of this section. Let $\mathcal{M} = \mathbb{F}_q^k$ be the space from which a file is picked and the encoding function $e : \mathcal{M} \to \mathcal{M}^*$, where $\mathcal{M}^* \subseteq \mathbb{F}_q^n$. Let $M = e(m)$ be the encoding of a file m. We define the challenge space to be $\{1, \ldots, n\}$ and the response function $\varrho(M, \textsc{Chal}) = M_{\textsc{Chal}}$ for a challenge $\textsc{Chal} \in \{1, \ldots, n\}$. We call this scheme the Basic Scheme.

Since the distance of the response code is d and the number of possible challenges is n, Theorem 4.1 gives us the following.

Corollary 4.2. *Suppose that $\mathcal{P}$ is a proving algorithm for the* Basic Scheme *for which* $\mathsf{succ}(\mathcal{P}) > 1 - d/(2n)$*, where the minimum Hamming distance of the set of encoded files* $\mathcal{M}^*$ *is* d *and M is an n-block encoded message. Then there is an* Extractor *that always outputs the file m.*

The basic scheme is very simple, but is not efficient in the sense that the verifier has to store as many encoded file blocks as the number of challenges it is going to make throughout the time when the file is stored on the server. Over the last few years, there have been considerable improvements made to the Basic Scheme. We cover some of them in more detail in Section 4.3.

4.3 Analysis of Several Keyless Schemes

In Section 4.2, we gave a general bound on the success probability of a proving algorithm that allows successful extraction of the file. In this section, we will use this as the basic recipe to analyze various PoR schemes.

4.3.1 PoR Code Construction of Dodis, Vadhan, and Wichs

We first analyze the "Basic PoR Code Construction" from Dodis, Vadhan, and Wichs [33] using the result in Theorem 4.1. This is a simple generalization of the basic scheme that we presented in the last section. We present the scheme in Figure 4.2. In this scheme, we have $\Delta = (\mathbb{F}_q)^\ell$.

Initialization

Given a *file* $m \in \mathcal{M}$, encode m as $e(m) = M \in \mathcal{M}^*$. The Verifier gives $M = (m_1, \ldots, m_n)$ to the Prover.

Challenge

A challenge is a subset of ℓ indices $J \subseteq \{1, \ldots, n\}$. Therefore, $\Gamma = \{J \subseteq \{1, \ldots, n\}, |J| = \ell\}$ and $\gamma = \binom{n}{\ell}$.

Response

Given the challenge $J = \{i_1, \ldots, i_\ell\}$ where $1 \leq i_1 < \ldots < i_\ell \leq n$, the correct response is the ℓ-tuple

$$\varrho(M, J) = (m_{i_1}, \ldots, m_{i_\ell}).$$

Suppose the Verifier receives a response $(r_1, \ldots, r_\ell)$. He then checks that $r_j = m_{i_j}$ for $1 \leq j \leq \ell$.

Figure 4.2: Multiblock Challenge Scheme

Lemma 4.3. *Let n be the block length of M and let every challenge in the* Multiblock Challenge Scheme *be an ℓ-tuple. Let the Hamming distance of $\mathcal{M}^*$ be d. Then the Hamming distance of the response code of the* Multiblock Challenge Scheme*, described in Figure 4.2, is $\mathsf{d}^* = \binom{n}{\ell} - \binom{n-\mathsf{d}}{\ell}$.*

Proof. Suppose that $M, M' \in \mathcal{M}^*$ and $M \neq M'$ such that $M = (m_1, \ldots, m_n)$ and $M' = (m'_1, \ldots, m'_n)$. Denote $\mathsf{dist}(M, M') = \delta$. It is easy to see that $\vec{r}_M = \vec{r}_{M'}$ if and only if $J \subseteq \{i : m_i = m'_i\}$, where $\vec{r}_M$ is as defined by Equation 4.1. From this, it is immediate that $\mathsf{dist}(\vec{r}_M, \vec{r}_{M'}) = \binom{n}{\ell} - \binom{n-\delta}{\ell}$. The desired result follows because $\delta \geq \mathsf{d}$. □

Now note that

$$1 - \frac{\mathsf{d}^*}{2\gamma} = 1 - \frac{\binom{n}{\ell} - \binom{n-\mathsf{d}}{\ell}}{2\binom{n}{\ell}} = \frac{1}{2} + \frac{\binom{n-\mathsf{d}}{\ell}}{2\binom{n}{\ell}}. \tag{4.2}$$

Combining Equation (4.2) with Lemma 4.3 and Theorem 4.1 gives us the following result.

Theorem 4.4. *Let n and ℓ be as defined in Lemma 4.3. Suppose that $\mathcal{P}$ is a proving algorithm for the* Multiblock Challenge Scheme *for which*

$$\mathsf{succ}(\mathcal{P}) > \frac{1}{2} + \frac{\binom{n-\mathsf{d}}{\ell}}{2\binom{n}{\ell}},$$

where the Hamming distance of $\mathcal{M}^*$ is d*. Then there is an* Extractor *that always outputs the file m.*

Remark 2. When we set $\ell = 1$ in Theorem 4.4, we obtain Corollary 4.2. □

4.3.2 Keyless Analogue of Shacham-Waters' Scheme

In this subsection, we consider the keyless analogue of the construction of Shacham and Waters [72]. We call the scheme the Linear Combination Scheme because a response consists of a specified linear combination of file blocks.

We illustrate the Linear Combination Scheme using the following example.

Example 11. Let $q = 11$. Let the message that the client wishes to store be $M = \big(4,\ \ 6,\ \ 3,\ \ 2\big)$. The Verifier picks a random challenge $V = \big(1,\ \ 0,\ \ 0,\ \ 1\big)$ and computes $\langle M, V\rangle = 6$. At later time, the Verifier sends the challenge V to which the Prover responds by computing $\langle M, V\rangle = 6$.

To apply Theorem 4.1, we need to compute the number of all possible challenges that can be made and the distance of the response code corresponding to the Linear Combination Scheme. The number of challenges can be computed easily: $\gamma = \binom{n}{\ell}(q-1)^{\ell}$. We need the following lemma to compute the distance of the response code.

Lemma 4.5. *Suppose that $\ell \geq 1$ and $X \in (\mathbb{F}_q)^{\ell}$ has Hamming weight equal to ℓ. Then the number of solutions $V \in (\mathbb{F}_q)^{\ell}$ to the equation $\langle V, X\rangle = 0$ in which V has Hamming weight equal to ℓ, which we denote by a_ℓ, is given by the formula*

$$a_\ell = \frac{q-1}{q}((q-1)^{\ell-1} - (-1)^{\ell-1}). \tag{4.3}$$

Proof. We prove the result by induction on ℓ.

Base Case: When $\ell = 1$, there are no solutions, so $a_1 = 0$, agreeing with Equation (4.3).

Induction: Now assume that Equation (4.3) gives the number of solutions for $\ell = s-1$, and consider $\ell = s$. Let $X = (x_1, \ldots, x_s)$ and define $X' = (x_1, \ldots, x_{s-1})$. By induction,

Initialization

Given a *file* $m \in \mathcal{M}$, encode m as $e(m) = M \in \mathcal{M}^*$. The Verifier gives $M = (M_1, \ldots, M_n)$ to the Prover.

Challenge

A challenge is an n-tuple $V = (v_1, \ldots, v_n) \in (\mathbb{F}_q)^n$ having Hamming weight ℓ.

Response

Given the challenge $V = (v_1, \ldots, v_n)$ of Hamming weight ℓ, the correct response is

$$\varrho(M, V) = \langle V, M \rangle,$$

where $M = (m_1, \ldots, m_n)$ and the computation is performed in $\mathbb{F}_q$. Suppose the Verifier receives a response $r \in \mathbb{F}_q$. He then checks that $r = \langle V, M \rangle$.

In this scheme, $\Delta = \mathbb{F}_q$.

Figure 4.3: Linear Combination Scheme

the number of solutions to the equation $\langle V', X' \rangle = 0$ in which V' has Hamming weight $s-1$ is a_{s-1}. Each of these solutions V' can be extended to a solution of the equation $\langle V, X \rangle = 0$ by setting $v_s = 0$; in each case, the resulting V has Hamming weight equal to $s-1$. However, any other vector V' of weight $s-1$ can be extended to a solution of the equation $\langle V, X \rangle = 0$ which has Hamming weight equal to s. Therefore, we have

$$\begin{aligned} a_s &= (q-1)^{s-1} - a_{s-1} \\ &= (q-1)^{s-1} - \left(\frac{q-1}{q}((q-1)^{s-2} - (-1)^{s-2}) \right) \\ &= (q-1)^{s-1} \left(1 - \frac{1}{q} \right) + \frac{q-1}{q}(-1)^{s-2} \\ &= \frac{q-1}{q}((q-1)^{s-1} - (-1)^{s-1}). \end{aligned}$$

□

We will now compute the distance d^* of the response code $\mathcal{M}^*$. Here is a lemma that will be of use in computing d^*.

Lemma 4.6. *Suppose that $M, M' \in \mathcal{M}^*$ and $M \neq M'$. Let n be the size of the message M and M'. Let ℓ be the Hamming weight of any challenge in the* Linear Combination Scheme*. Denote $\delta = \mathsf{dist}(M, M')$. Let $\vec{r}_M$ and $\vec{r}_{M'}$ be the corresponding vectors in the response code of the* Linear Combination Scheme*. Then*

$$\mathsf{dist}(\vec{r}_M, \vec{r}_{M'}) = (q-1)^\ell \left(\binom{n}{\ell} - \binom{n-\delta}{\ell} \right) - \sum_{w \geq 1} \binom{\delta}{w} \binom{n-\delta}{\ell - w} (q-1)^{\ell - w} a_w, \tag{4.4}$$

where the a_w's are given by Equation (4.3).

Proof. Suppose that $M, M' \in \mathcal{M}^*$ and $M \neq M'$. Denote

$$M = (m_1, \ldots, m_n) \text{ and } M' = (m'_1, \ldots, m'_n),$$

and let $\delta = \mathsf{dist}(M, M')$. Let

$$J = \{i : m_i = m'_i\} \text{ and } J' = \{1, \ldots, n\} \setminus J.$$

Observe that $|J| = n - \delta$ and $|J'| = \delta$. For any $V = (v_1, \ldots, v_n)$ having Hamming weight equal to ℓ, define

$$J_V = \{j \in J : v_j \neq 0\} \text{ and } J'_V = \{j \in J' : v_j \neq 0\}.$$

Denote $w = |J'_V|$; then $|J_V| = \ell - w$.

Suppose $w \geq 1$. Then, given J_V and J'_V, the number of solutions to the equation $\langle V, M \rangle = \langle V, M' \rangle$ is precisely $(q-1)^{\ell - w} a_w$. When $w = 0$, the number of solutions is $(q-1)^\ell$. Summing over w, and considering all possible choices for J_V and J'_V, we have that the total number of solutions to the equation $\langle V, M \rangle = \langle V, M' \rangle$ is

$$\binom{n-\delta}{\ell} (q-1)^\ell + \sum_{w \geq 1} \binom{\delta}{w} \binom{n-\delta}{\ell - w} (q-1)^{\ell - w} a_w. \tag{4.5}$$

The desired result follows. □

We can obtain a very accurate estimate for d^* by observing that

$$a_w \approx \frac{(q-1)^w}{q}$$

is a very accurate approximation. After making this substitution, it is easy to see that Equation (4.5) is minimised when $\delta = \mathsf{d}$. This minimises Equation (4.4), so we obtain

$$\mathsf{d}^* \approx \binom{n}{\ell}(q-1)^\ell - \binom{n-\mathsf{d}}{\ell}(q-1)^\ell - \sum_{w \geq 1} \binom{d}{w}\binom{n-\mathsf{d}}{\ell - w}\frac{(q-1)^\ell}{q}. \tag{4.6}$$

We have

$$\begin{aligned} \sum_{w \geq 1} \binom{d}{w}\binom{n-\mathsf{d}}{\ell - w}\frac{(q-1)^\ell}{q} &= \frac{(q-1)^\ell}{q}\sum_{w \geq 1} \binom{d}{w}\binom{n-\mathsf{d}}{\ell - w} \\ &= \frac{(q-1)^\ell}{q}\left(\binom{n}{\ell} - \binom{n-\mathsf{d}}{\ell}\right), \end{aligned}$$

where the last equality follows from Graham, Knuth, and Patashnik [44]. Therefore, from Equation (4.6), we get

$$\begin{aligned} \mathsf{d}^* &\approx (q-1)^\ell\left(\binom{n}{\ell} - \binom{n-\mathsf{d}}{\ell}\right) - \frac{(q-1)^\ell}{q}\left(\binom{n}{\ell} - \binom{n-\mathsf{d}}{\ell}\right) \\ &= \frac{(q-1)^{\ell+1}}{q}\left(\binom{n}{\ell} - \binom{n-\mathsf{d}}{\ell}\right). \end{aligned} \tag{4.7}$$

The following theorem uses the estimated value for d^* derived in Equation (4.7).

Theorem 4.7. *Let n and ℓ be as defined in Lemma 4.6. Suppose that $\mathcal{P}$ is a proving algorithm of the* Linear Combination Scheme *for which*

$$\mathsf{succ}(\mathcal{P}) \gtrsim \frac{1}{2} + \frac{1}{2}\left(\frac{1}{q} + \frac{(q-1)\binom{n-\mathsf{d}}{\ell}}{q\binom{n}{\ell}}\right),$$

where the Hamming distance of $\mathcal{M}^*$ is d. Then there exists an Extractor *which will always output the file m.*

4.4 The Shacham-Waters Scheme

We now turn our attention to the Shacham-Waters prototype [72] which is a *keyed proof-of-retrievability scheme*. The term "keyed" means that the Verifier has a secret key that is not provided to the Prover. This key is used to verify responses in the challenge-response

- The *key* K consists of $a \in \mathbb{F}_q$ and $B = (b_1, \ldots, b_n) \in (\mathbb{F}_q)^n$. K is retained by the Verifier.
- The *encoded file* is $M = (m_1, \ldots, m_n) \in (\mathbb{F}_q)^n$.
- The *tag* is $S = (\sigma_1, \ldots, \sigma_n) \in (\mathbb{F}_q)^n$, where S is computed using the following (vector) equation in $\mathbb{F}_q$:
$$S = B + aM. \tag{4.8}$$
The file M and the tag S are given to the Prover.
- A *challenge* is a vector $V = (v_1, \ldots, v_n) \in (\mathbb{F}_q)^n$.
- The *response* consists of $(\textsc{Resp}_1, \textsc{Resp}_2) \in (\mathbb{F}_q)^2$, where the following computations are performed in $\mathbb{F}_q$:
$$\textsc{Resp}_1 = \langle V, M \rangle \tag{4.9}$$
and
$$\textsc{Resp}_2 = \langle V, S \rangle. \tag{4.10}$$
- The response $(\textsc{Resp}_1, \textsc{Resp}_2)$ is *verified* by checking that the following condition holds in $\mathbb{F}_q$:
$$\textsc{Resp}_2 \overset{?}{=} a\textsc{Resp}_1 + \langle V, B \rangle. \tag{4.11}$$

Figure 4.4: Modified Shacham-Waters Scheme

protocol, and it is also provided to an extraction algorithm as an input. The use of a key permits an arbitrary number of challenges to be verified, without the Verifier having to precompute the responses.

The construction of Shacham and Waters [72] is insecure in the unconditional security framework. This is because a computationally unbounded adversary can simply break the underlying hardness assumption, which in this case leads to to the scheme being broken. Therefore, for any non-trivial result, we have to modify their scheme. One of the modifications is presented in Figure 4.4. The main change is that the vector $B := (b_1, \ldots, b_n)$ is completely random instead of being generated by a pseudorandom function. We call this scheme the Modified Shacham-Waters Scheme.

We give an example to illustrate the Modified Shacham-Waters Scheme.

Example 12. Let $q = 11$, $a = 2, B = \begin{pmatrix} 2, & 4, & 6, & 7 \end{pmatrix}$. Let the message that the client wishes to store be $M = \begin{pmatrix} 4, & 6, & 3, & 2 \end{pmatrix}$. Then the corresponding tag is $S = B + aM = \begin{pmatrix} 10, & 5, & 1, & 0 \end{pmatrix}$. The file $\begin{pmatrix} 4, & 6, & 3, & 2 \end{pmatrix}$ and the tag $\begin{pmatrix} 10, & 5, & 1, & 0 \end{pmatrix}$ is stored on the Prover. During an audit phase, the Verifier picks a random challenge $\begin{pmatrix} 1, & 0, & 0, & 1 \end{pmatrix}$. The Prover computes $\textsc{Resp}_1 = \langle M, V \rangle = 6$ and $\textsc{Resp}_2 = \langle S, V \rangle = 10$. The Verifier verifies by computing $a\textsc{Resp}_1 + \langle V, B \rangle = 12 + 9 \equiv 10 \mod 11 = \textsc{Resp}_2$.

There is an asymmetry in the terms of information held by various entities in the scheme: the Verifier and Extractor have the secret key, while the Extractor also has access to the proving algorithm $\mathcal{P}$ generated by the Prover. The Prover has access to the file-tag pair (M, S). We tabulate the information held by different entities in the modified Shacham-Waters prototype below for the ease of reference.

Entities	Verifier	Prover	Extractor
Information held	$K = (a, B)$	M, S	$K, \mathcal{P}$

From the point of view of the Prover, there are q possible keys. Suppose the first tuple of the key is $a = a_0$. Then Equation (4.8) implies that $B = S - a_0 M$. In other words, we have the following.

Lemma 4.8. *Given M and S, the* Prover *can restrict the set of possible keys (a, B) in the Modified Shacham-Waters Scheme, described in Figure 4.4, to*

$$\mathsf{Possible}(M, S) = \{(a_0, S - a_0 M) : a_0 \in \mathbb{F}_q\}.$$

We say a response $(\textsc{Resp}_1, \textsc{Resp}_2)$ is *acceptable* if Equation (4.11) is satisfied and *authentic* if it was created properly using Equations (4.9) and (4.10). Note that it is possible for a response to be acceptable but not authentic because the set of acceptable responses contains the set of authentic responses as a subspace. On the other hand, an authentic response will be acceptable for every key $K \in \mathsf{Possible}(M, S)$. In the case of an acceptable (but perhaps not authentic) response, we have the following useful lemma.

Lemma 4.9. *Suppose that a response $(\textsc{Resp}_1, \textsc{Resp}_2)$ to a challenge V for a file M is acceptable for more than one key in $\mathsf{Possible}(M, S)$ in the* Modified-Shacham-Waters Scheme*. Then $(\textsc{Resp}_1, \textsc{Resp}_2)$ is authentic.*

Proof. Suppose $K_1, K_2 \in \mathsf{Possible}(M, S)$, where $K_1 = (a_1, B_1)$, $K_2 = (a_2, B_2)$ and $a_1 \neq a_2$. We have $B_1 = S - a_1 M$ and $B_2 = S - a_2 M$. Now consider a response $(\textsc{Resp}_1, \textsc{Resp}_2)$ to a challenge V that is acceptable for both of the keys K_1 and K_2. Then

$$\textsc{Resp}_2 = a_1 \textsc{Resp}_1 + \langle V, B_1 \rangle = a_2 \textsc{Resp}_1 + \langle V, B_2 \rangle.$$

Therefore,

$$(a_1 - a_2)\textsc{Resp}_1 + \langle V, (B_1 - B_2)\rangle = 0.$$

However, we have $B_1 - B_2 = (a_2 - a_1)M$, so

$$(a_1 - a_2)(\textsc{Resp}_1 - \langle V, M\rangle) = 0.$$

We have $a_1 \neq a_2$, so it follows that $\textsc{Resp}_1 = \langle V, M\rangle$. Then we obtain

$$\begin{aligned}\textsc{Resp}_2 &= a_1\textsc{Resp}_1 + \langle V, B_1\rangle \\ &= a_1\langle V, M\rangle + \langle V, (S - a_1 M)\rangle \\ &= \langle V, S\rangle.\end{aligned}$$

Therefore the response $(\textsc{Resp}_1, \textsc{Resp}_2)$ is authentic. □

We first state the reasons why the keyed PoR scheme presented in Figure 4.4 is not secure in the standard security definition of PoR systems.

Towards Defining the Average-case Security. We start our argument with a simple attack on the Shacham-Waters' construction using a computationally unbounded adversary. The crucial observation which facilitates our attack is that the verification condition (4.11) depends on the key, but not on the file M. As a result, the Prover can create acceptable non-authentic responses if he knows the value of a. It suffices to know the value of a to break the construction of Shacham and Waters [72] due to Lemma 4.8. Our attack is based on the following theorem.

Theorem 4.10. *If the Prover has access to a verification oracle, then the Modified Shacham-Waters Scheme is not unconditionally secure.*

Proof. For every key $K \in \mathsf{Possible}(M, S)$, the Prover can create a response $(\textsc{Resp}_1, \textsc{Resp}_2)$ to a challenge V that will be acceptable if and only if K is the actual key (this follows from Lemma 4.9). The Prover can check the validity of these responses by accessing the verification oracle. As soon as one of these responses is accepted by the verification oracle, the Prover knows the correct value of the key. Hence the Prover can now create a proving algorithm $\mathcal{P}$ that will output acceptable but non-authentic responses. This algorithm $\mathcal{P}$ will not allow the correct file to be extracted.

In more detail, after the Prover has determined the key $K = (a, B)$, he chooses an arbitrary (encoded) file $M' \neq M$ and constructs $\mathcal{P}$ as follows:

1. Given a challenge V, define $\textsc{Resp}_1 = \langle V, M' \rangle$.
2. Define $\textsc{Resp}_2 = a\textsc{Resp}_1 + \langle V, B \rangle$.

Suppose the Extractor is run on $\mathcal{P}$. It is easy to see that $\mathsf{dist}(R', \vec{r}_{M'}) = 0$, so the Extractor will compute $\widehat{M} = M'$, which is incorrect. □

Now suppose we modify the definition of PoR systems by removing the access to verification oracle by the Prover. It is straightforward to see that even in the absence of a verification oracle, the Prover can guess the correct value of a with probability $1/q$. If the adversary correctly guess the value of a, then it can create a non-extractable proving algorithm. This implies that it is not possible to prove a theorem stating that *any* proving algorithm yields an extractor.

The above discussion seems to present a pessimistic picture of what can and what cannot be achieved by a keyed PoR system in the setting of unconditional security. However, we can prove meaningful reductions if we define the success probability of a proving algorithm to be the *average* success probability over the q possible keys that are consistent with the information given to the Prover. We next define this idea more formally.

Average-case Success Probability of a Prover. Suppose $\mathcal{P}$ is a (deterministic) proving algorithm for a file $M = (m_1, \ldots, m_n) \in (\mathbb{F}_q)^n$. For each challenge V and for every key $K = (a, B) \in \mathsf{Possible}(M, S)$, define

$$\chi(V, K) := \begin{cases} 1 & \text{if } \mathcal{P} \text{ returns an acceptable response for the key } K \text{ given the challenge } V, \\ 0 & \text{otherwise.} \end{cases}$$

Since there are $\gamma q = q^{n+1}$ choices for the pair (V, K), we define the *average success probability* $\mathsf{succ}_{\mathsf{avg}}(\mathcal{P})$ to be

$$\mathsf{succ}_{\mathsf{avg}}(\mathcal{P}) = \frac{\displaystyle\sum_{V \in \Gamma, K \in \mathsf{Possible}(M,S)} \chi(V, K)}{\gamma q}. \tag{4.12}$$

Lemma 4.11. *Let q and γ be as defined above. Suppose there are d challenges V for which $\mathcal{P}$ returns an authentic response, and hence there are $C = \gamma - \mathsf{d}$ challenges for which $\mathcal{P}$ returns a response that is not authentic. Then*

$$\mathsf{succ}_{\mathsf{avg}}(\mathcal{P}) \leq 1 - \frac{C(q-1)}{\gamma q}. \tag{4.13}$$

Proof. If V is a challenge for which $\mathcal{P}$ returns an authentic response, then $\chi(V, K) = 1$ for every K. If V is a challenge for which $\mathcal{P}$ does not return an authentic response, then Lemma 4.9 implies that $\chi(V, K) = 1$ for at most one K. Therefore,

$$\sum_{V \in \Gamma, K \in \mathsf{Possible}(M,S)} \chi(V, K) \leq C + qD = \gamma q - C(q-1).$$

The desired result now follows from (4.12). □

Let us now turn our attention to the response code. Using Equation (4.11), we know that $\textsc{Resp}_1$, K and V uniquely determine $\textsc{Resp}_2$; therefore, it suffices to consider only the values of $\textsc{Resp}_1$ in the extraction process. We define the response vector for a file M to be $\vec{r}_M = (\langle M, V \rangle : V \in \Gamma)$. Observe that this response vector is identical to the response vector in the Linear Combination Scheme.

Lemma 4.12. *Suppose that $\mathcal{P}$ is a proving algorithm for the file M in the* Modified Shacham-Waters Scheme*. Let $\vec{r}_M = (\langle M, V \rangle : V \in \Gamma)$ and let R' be the γ-tuple of responses computed by $\mathcal{P}$. Then*

$$\mathsf{dist}(\vec{r}_M, R') \leq \frac{(1 - \mathsf{succ}_{\mathsf{avg}}(\mathcal{P}))\gamma q}{q-1} = \frac{(1 - \mathsf{succ}_{\mathsf{avg}}(\mathcal{P}))q^{n+1}}{q-1}.$$

Proof. Define C as in Lemma 4.11. Since a co-ordinate of $\vec{r}_M$ differs from the corresponding co-ordinate of R' only when the response is non-authentic, it follows that $\mathsf{dist}(\vec{r}_M, R') \leq C$. Equation (4.13) implies that

$$C \leq \frac{(1 - \mathsf{succ}_{\mathsf{avg}}(\mathcal{P}))\gamma q}{q-1},$$

from which the stated result follows. □

We now present our security result for the Modified Shacham-Waters Scheme.

Theorem 4.13. *Let q be the size of the underlying field used in the* Modified Shacham-Waters Scheme*. Suppose that*

$$\mathsf{succ}_{\mathsf{avg}}(\mathcal{P}) > 1 - \frac{\mathsf{d}^*(q-1)}{2\gamma q}, \tag{4.14}$$

where d^* is given by Equation (4.4) and $\gamma = q^n$. Then there exists an Extractor *that always outputs $\widehat{m} = m$.*

Proof. Denote $\eta = \mathsf{succ}_{\mathsf{avg}}(\mathcal{P})$, let R' be the γ-tuple of responses computed by $\mathcal{P}$, and denote $\delta = \mathsf{dist}(\vec{r}_M, R')$, where $M = e(m)$.

We first present the extractor.

1. On input $\mathcal{P}$, compute the vector $R' = (\textsc{Resp}_{1,V} : V \in \Gamma)$, where $(\textsc{Resp}_{1,V}, \textsc{Resp}_{2,V}) = \mathcal{P}(V)$ for all $V \in \Gamma$.

2. Find $\widehat{M} \in \mathcal{M}^*$ so that $\mathsf{dist}(R', \vec{r}_{\widehat{M}})$ is minimised.

3. Output $\widehat{m} = e^{-1}(\widehat{M})$.

We showed in Lemma 4.12 that

$$\delta \leq \frac{(1-\eta)\gamma q}{q-1}.$$

We want to prove that $\widehat{M} = M$. We have that $\vec{r}_{\widehat{M}}$ is a codeword in $\mathcal{R}^*$ closest to R'. Since M is a codeword such that $\mathsf{dist}(\vec{r}_M, R') = \delta$, it must be the case that $\mathsf{dist}(\vec{r}_{\widehat{M}}, R') \leq \delta$. By the triangle inequality, we get

$$\mathsf{dist}(\vec{r}_M, \vec{r}_{\widehat{M}}) \leq \mathsf{dist}(\vec{r}_M, R') + \mathsf{dist}(\vec{r}_{\widehat{M}}, R') \leq \delta + \delta = 2\delta.$$

However,

$$2\delta \leq \frac{2(1-\eta)\gamma q}{q-1} < \mathsf{d}^*,$$

where the last inequality follows from (4.14). Since $\vec{r}_M$ and $\vec{r}_{\widehat{M}}$ are codewords within distance d^* (which is the distance of the response code), it follows that $M = \widehat{M}$ and the $\mathsf{Extractor}$ outputs $m = e^{-1}(M)$, as desired. □

4.5 Estimating the Success Probability of a Prover

The essential purpose of a PoR scheme is to assure the user that their file is indeed being stored correctly; i.e., in such a manner that the user can recover the entire file if desired. We considered several schemes for testing whether this is the case or not and gave a bound that allows successful recovery of the entire file. However, in order to interpret these results in practice, there are two important issues with these results. The first is that the presence of binomial coefficients makes the expressions of the minimum success probability required for successful extraction in all three schemes somewhat complicated. The second issue is that, in practice, one is more likely to do only a small number of audits, and, therefore, it is not obvious how can one be assured that the success probability of the proving algorithm is high enough to allow successful extraction. We address these two issues in the remainder of this section.

4.5.1 Numerical Computations and Estimates

We have provided sufficient conditions for extraction to succeed for several PoR schemes, based on the success probability of the proving algorithm. Here, we look a bit more closely at these numerical conditions and provide some useful comparisons and estimates for the different schemes we have studied. The following expressions are straightforward.

1. For the Multiblock Challenge Scheme, Theorem 4.4 guarantees that extraction will succeed if
$$\mathsf{succ}(\mathcal{P}) > \mathsf{Succ}_0 = \frac{1}{2} + \frac{\binom{n-\mathsf{d}}{\ell}}{2\binom{n}{\ell}}.$$

2. For the Linear Combination Scheme, Theorem 4.7 guarantees that extraction will succeed if
$$\mathsf{succ}(\mathcal{P}) > \mathsf{Succ}_1 = \left(\frac{q-1}{q}\right)\mathsf{Succ}_0 + \frac{1}{q}.$$

3. For the Modified Shacham-Waters Scheme, Theorem 4.13, using the estimate for d^* given in Equation (4.7), guarantees that extraction will succeed if
$$\mathsf{succ}_{\mathsf{avg}}(\mathcal{P}) > \mathsf{Succ}_2 = \left(\frac{q-1}{q}\right)^2\mathsf{Succ}_0 + \frac{2}{q} - \frac{1}{q^2}.$$

It is clear that $\mathsf{Succ}_0, \mathsf{Succ}_1$ and Succ_2 are extremely close for any reasonable value of q (such as $q \geq 2^{32}$, for example). Therefore we will confine our subsequent analysis to S_0 and state our results in terms of the Multiblock Challenge Scheme. The following theorem gives a approximate estimate of Succ_0, which is far easier to verify rather than the expression Succ_0.

Theorem 4.14. *Denote $\eta' = 1 - \mathsf{succ}(\mathcal{P})$. Suppose that the following inequality holds in the* Multiblock Challenge Scheme*:*

$$\frac{\ell d}{n} > \ln\left(\frac{1}{1-2\eta'}\right). \tag{4.15}$$

Then the Extractor *will always succeed.*

Proof. From Equation (4.15), we obtain

$$\ln(1-2\eta') > -\frac{\ell\mathsf{d}}{n}.$$

Now $-x > \ln(1-x)$ for $0 < x < 1$, we obtain

$$\ln(1-2\eta') > \ell \ln\left(1 - \frac{\mathsf{d}}{n}\right).$$

Exponentiating both sides of this inequality, we have

$$1 - 2\eta' > \left(1 - \frac{\mathsf{d}}{n}\right)^{\ell}.$$

A simple calculation shows that

$$\left(1 - \frac{\mathsf{d}}{n}\right)^{\ell} \geq \frac{\binom{n-\mathsf{d}}{\ell}}{\binom{n}{\ell}},$$

so it follows that

$$1 - 2\eta' > \frac{\binom{n-\mathsf{d}}{\ell}}{\binom{n}{\ell}}.$$

From this, we obtain

$$\mathsf{succ}(\mathcal{P}) > \frac{1}{2} + \frac{\binom{n-\mathsf{d}}{\ell}}{2\binom{n}{\ell}},$$

and hence the Extractor will always succeed. □

Table 4.1 lists values of n (the length of an encoded file), for different values of ℓ (the Hamming weight of the challenge), d (the distance of the code) and the success probability of the proving algorithm, such that the Extractor is guaranteed to succeed. We list the value of n as specified by Theorems 4.4 and 4.14. We see, for a wide range of parameters, that the estimate obtained in Theorem 4.14 is very close to the earlier value computed in Theorem 4.4.

Table 4.1: Values of n for which the Extractor will always succeed

ℓ	d	succ($\mathcal{P}$)	n (Thm. 4.4)	n (Thm. 4.14)
10000	10000	0.6	62143493	62133493
		0.7	109145666	109135666
		0.8	195771518	195761518
		0.9	448152011	448142011
		0.99	4949841645	4949831645
10000	1000	0.6	6218850	6213349
		0.7	10919066	10913566
		0.8	19581651	19576151
		0.9	44819701	44814201
		0.99	494988664	494983164
10000	100	0.6	626398	621334
		0.7	1096413	1091356
		0.8	1962668	1957615
		0.9	4486471	4481420
		0.99	4950336	49498316
10000	10	0.6	67272	62133
		0.7	114216	109136
		0.8	200808	195761
		0.9	453165	448142
		0.99	4954838	4949832
1000	10000	0.6	6218850	6213349
		0.7	10919066	10913567
		0.8	19581651	19576152
		0.9	44819700	44814201
		0.99	494988664	494983164
1000	1000	0.6	622334	6213349
		0.7	1092356	10913567
		0.8	1958614	19576152
		0.9	4482419	4481420
		0.99	49499315	49498316
1000	100	0.6	62684	62133
		0.7	109685	109135

Continued on next page . . .

Table 4.1 — continued from previous page

ℓ	d	succ($\mathcal{P}$)	n (Thm. 4.4)	n (Thm. 4.14)
		0.8	196311	195761
		0.9	448692	448142
		0.99	4950381	4949831
1000	10	0.6	6731	6213
		0.7	11425	10914
		0.8	20084	19576
		0.9	45320	44814
		0.99	495488	494983
100	10000	0.6	626398	621334
		0.7	1096413	1091357
		0.8	1962669	1957615
		0.9	4486471	4481420
		0.99	49503366	49498316
100	1000	0.6	62684	62133
		0.7	109685	109135
		0.8	196311	195761
		0.9	448691	448142
		0.99	4950381	4949831
100	100	0.6	6313	6213
		0.7	11013	10913
		0.8	19675	19576
		0.9	44913	44814
		0.99	495082	494983
100	10	0.6	677	621
		0.7	1146	1091
		0.8	2012	1958
		0.9	4536	4481
		0.99	49552	49498
50	10000	0.6	315719	310667
		0.7	550718	5456783
		0.8	983840	9788076
		0.9	2245736	2240710
		0.99	24754183	24749158

Continued on next page ...

Table 4.1 — continued from previous page				
ℓ	d	$\mathsf{succ}(\mathcal{P})$	n (Thm. 4.4)	n (Thm. 4.14)
50	1000	0.6	31594	31068
		0.7	55093	545678
		0.8	98406	978807
		0.9	224599	224071
		0.99	2475440	2474916
50	100	0.6	3181	3106
		0.7	5531	5456
		0.8	9862	9788
		0.9	22481	22407
		0.99	247565	247492
50	10	0.6	341	311
		0.7	576	546
		0.8	1009	979
		0.0	2270	2240
		0.99	24779	24749

4.5.2 Statistical Techniques for Estimating Success Probabilities in PoR Systems

Our main result can be interpreted as saying that extraction is possible for any keyless scheme whenever $\mathsf{succ}(\mathcal{P})$ is at least $1-\tilde{\mathsf{d}}/n$. Hence, the information we would like to obtain from using the Multiblock Challenge Scheme or the Linear Combination Scheme is whether we can compute a value ω such that extraction is possible whenever $\mathsf{succ}(\mathcal{P}) > \frac{\omega-1}{\gamma}$. We can calculate $\mathsf{succ}(\mathcal{P})$ for a given proving algorithm $\mathcal{P}$ if we know the values of the proving algorithm's response $\mathcal{P}(c)$ for every possible challenge $c \in \Gamma$.

On the other hand, from the practical point of view, the whole purpose of a PoR scheme is to provide reassurance that $\mathsf{succ}(\mathcal{P})$ is sufficiently large without having to request $\mathcal{P}(c)$ for all $c \in \Gamma$. Given the prover's responses to some subset of possible challenges, the verifier wishes to make a judgement as to whether $\mathsf{succ}(\mathcal{P})$ is acceptably high.

Hypothesis Testing

One of the standard statistical techniques which is used in the scenario as described above is *hypothesis testing* [22, 28]. The general method of hypothesis testing can be summarized in four steps:

1. We identify a hypothesis that we feel should be tested.
2. We select a criteria for deciding whether the hypothesis is true or not.
3. We select a random sample from the whole sample space and measure the outcome of the test on the sample.
4. We compare the observed values to what we expect if the hypothesis is true.

Returning to our scenario of testing whether $\mathsf{succ}(\mathcal{P})$ is high enough, we set up the system of hypothesis testing as follows. Let $c_1, \ldots, c_\tau$ be the random challenges made by $\mathsf{Verifier}$. Let $\varrho_1, \ldots, \varrho_\tau$ be the responses of $\mathcal{P}$, and suppose that $\mathsf{Verifier}$ accepts g of these challenges. Since we assume that the Prover is an adversary, we are concerned that $\mathsf{succ}(\mathcal{P})$ is not high enough to allow successful extraction. Therefore, our goal is to provide evidence that convinces the verifier that $\mathsf{succ}(\mathcal{P})$ is sufficiently high to allow successful extraction. In the terminology of *hypothesis testing*, this means that we wish to distinguish the *null hypothesis*

$\mathsf{Hyp}_0 : \mathsf{succ}(\mathcal{P}) \leq \frac{\omega-1}{\gamma}$;

from the *alternative hypothesis*

$\mathsf{Hyp}_1 : \mathsf{succ}(\mathcal{P}) \geq \frac{\omega}{\gamma}$.

In hypothesis testing, we assume that the null hypothesis is true and try to find evidence that rebuts the null hypothesis. In this setting, we state the *level of significance* for a test, often called the *p-value*. Usually, the level of significance is based on certain empirical rules. In statistical hypothesis testing, it is generally accepted that 5% is a good level of significance. In other words, if $p \leq 0.05$, we reject the null hypothesis; otherwise, we continue to retain the null hypothesis.

In our case, we first suppose that Hyp_0 is true. Since the challenges are picked without replacement, the probability that the number of correct responses is at least g is at most

$$\sum_{i=g}^{t} \frac{\binom{\omega-1}{i}\binom{\gamma-\omega+1}{t-i}}{\binom{\gamma}{t}}. \tag{4.16}$$

If the expression (4.16) representing the probability is less than 0.05, then we reject Hyp_0 and instead accept the alternative hypothesis (namely that $\mathsf{succ}(\mathcal{P})$ is sufficiently high to permit extraction). In this case we conclude that the server is storing the file appropriately. If the probability in Equation (4.16) is greater than 0.05, then there is insufficient evidence to reject Hyp_0 at the 5% significance level. Therefore, we continue to suspect that the server's proving algorithm would not permit successful extraction.

Now consider the case when the challenges are picked uniformly at random *with* replacement. Under this condition for rejecting the null hypothesis becomes

$$\sum_{i=g}^{t} \binom{t}{i} \left(\frac{\omega - 1}{\gamma}\right)^{i} \left(\frac{\gamma - \omega + 1}{\gamma}\right)^{t-i} < 0.05.$$

We illustrate the above discussion with a help of following example.

Example 13. Consider the Basic Scheme instantiated with $n = 1000$ blocks. Let the distance of the response code be 400. Then Corollary 4.2 gives that extraction is possible whenever $\mathsf{succ}(\mathcal{P})$ is greater than 0.8. Suppose the verifier invokes a total of 100 challenge-response protocols with the challenges chosen uniformly at random with replacement. For the sake of illustration, let us assume that the Verifier accepts a total of 87 audits. A simple calculation shows that

$$\sum_{i=87}^{100} \binom{100}{i} (0.8)^{i} (0.2)^{100-i} \approx 0.047 < 0.05.$$

The above illustration shows that, in the above case, there is sufficient evidence to reject the null hypothesis at the 5% significance level, and so we believe that the file is in fact being stored correctly.

Now suppose, on the contrary, that if only 86 of the responses were correct, then

$$\sum_{i=86}^{100} \binom{100}{i} (0.8)^{i} (0.2)^{100-i} \approx 0.08 > 0.05.$$

In this case there is not enough evidence to reject the null hypothesis at the 5% significance level, and so we believe that the server?s proving algorithm would permit successful extraction.

The benefit of this statistical approach is that, given the observed responses to the challenges, for any desired value of a we can construct a hypothesis test for which the probability of inappropriately rejecting the null hypothesis (and hence failing to catch a prover that does not permit extraction) is necessarily less than a.[1] This is the case regardless of the true value of $\mathsf{succ}(\mathcal{P})$, and we do not need to make any *a priori* assumptions about this value.

In Table 4.2 we give examples of a range of possible results of the challenge process and the corresponding outcomes in terms of whether the null hypothesis is rejected at either the 5% or 1% significance level. The columns headed by values of a contain a ick if Hyp_0 is rejected at the corresponding significance level, and a cross otherwise.

Confidence Intervals

Another closely related way to portray the information provided by the sample of responses to challenges is through the use of confidence intervals. We define a 95% *lower confidence bound* θ_L by

$$\theta_L = \sup\left\{\theta \middle| \sum_{i=g}^{t} \binom{t}{i} \theta^i (1-\theta)^{t-i} < 0.05\right\}$$

The above expression gives the largest possible value for $\mathsf{succ}(\mathcal{P})$ for which the probability of obtaining g or more correct responses in a sample of size t is less than 0.05. In other words, θ_L signifies that whenever $\frac{\omega-1}{\gamma} < \theta_L$, the probability of a prover with success rate at most $\frac{\omega-1}{\gamma}$ providing g or more correct responses is less than 0.05. Therefore, the decision process for the hypothesis test described in Section 4.5.2 consists of rejecting the null hypothesis whenever $\frac{\omega-1}{\gamma} < \theta_L$.

The interval $(\theta_L, 1]$ is a 95% *confidence interval* for $\mathsf{succ}(\mathcal{P})$: if a large number of samples of size t were made and the corresponding intervals were calculated using this approach, then one would expect the resulting intervals to contain the true value of $\mathsf{succ}(\mathcal{P})$ at least 95% of the time. The hypothesis test can be expressed in terms of the confidence interval $(\theta_L, 1]$ by stating that we reject Hyp_0 whenever $\frac{\omega-1}{\gamma}$ does not lie in this interval.

Example 14. Suppose we have $n = 1000$ and $d = 400$ as in Example 13, and suppose that 90 of the responses are correct. Then

$$\begin{aligned}\theta_L &= \sup\left\{\theta \middle| \sum_{i=90}^{t} \binom{t}{i} \theta^i (1-\theta)^{t-i} < 0.05\right\} \\ &\approx 0.836.\end{aligned}$$

[1] Here we refer to the probability over the set of all possible choices of t challenges.

Table 4.2: Outcomes of Hypothesis Testing for a Range of Responses.

$\frac{\omega-1}{\gamma}$	t	g	$a = 0.05$	$a = 0.01$	$\frac{\omega-1}{\gamma}$	t	g	$a = 0.05$	$a = 0.01$
0.8	100	100	✓	✓	0.9	100	100	✓	✓
0.8	100	95	✓	✓	0.9	100	95	X	X
0.8	100	90	X	X	0.9	100	90	X	X
0.8	100	85	X	X	0.9	100	85	X	X
0.8	100	80	X	X	0.9	100	80	X	X
0.8	200	180	✓	✓	0.9	200	200	✓	✓
0.8	200	175	✓	✓	0.9	200	195	✓	✓
0.8	200	170	✓	X	0.9	200	190	✓	✓
0.8	200	165	X	X	0.9	200	185	X	X
0.8	200	160	X	X	0.9	200	180	X	X
0.8	500	435	✓	✓	0.9	500	480	✓	✓
0.8	500	430	✓	✓	0.9	500	475	✓	✓
0.8	500	425	✓	✓	0.9	500	470	✓	✓
0.8	500	420	✓	X	0.9	500	465	✓	X
0.8	500	415	X	X	0.9	500	460	X	X

Then a 95% confidence interval for $\mathsf{succ}(\mathcal{P})$ is $(0.836, 1]$, and hence we reject the null hypothesis, as $\frac{\omega-1}{n} = 0.8$ does not lie in this interval.

One question that has not usually been considered is what action to take when a prover is suspected of cheating. In the framework of Section 4.5.2, this becomes the problem of what to do in the case where there is insufficient evidence to reject the null hypothesis. There are various possible options at this point, and the choice of option will depend on factors such as the reason for storing the file, and any costs and inconvenience that might be associated with ceasing to use that server, or with switching to another storage provider. For example, if a server is simply being used as a backup service for non-critical data and there is a high overhead associated with switching storage providers, then a user will not want to be overhasty in taking action against a possibly innocent server. In this case, an appropriate action in the first instance might be to seek more responses to challenges in order to avoid the possibility that the earlier set of responses were unrepresentative of the reliability of the prover in general.

Comparison with Approaches Followed in the Literature. The use of hypothesis testing and confidence intervals has not been used explicitly in the literature pertaining to PoR schemes. In this section, we cast some of the earlier analyses in terms of hypothesis testing. Shacham and Waters [72] showed how the main construction of Ateniese *et al.* [3] could be turned into a PoR scheme by demonstrating an extractor which extracts the file, given a good enough proving algorithm. The following analysis was done by Ateniese *et al.* [3]. The authors observed that if $\mathsf{succ}(\mathcal{P}) \leq \frac{g}{\gamma}$ (with $g \in \mathbb{Z}$) then, when $\mathcal{P}$ is queried on t possible challenges chosen uniformly at random without replacement, the probability p that at least one incorrect response is observed is given by

$$\mathsf{p} = 1 - \frac{\binom{g}{t}}{\binom{\gamma}{t}},$$

and they note that

$$1 - \left(\frac{g}{\gamma}\right)^t \leq \mathsf{p} \leq 1 - \left(\frac{g+1-t}{\gamma - t + 1}\right)^t .$$

Ateniese *et al.* [3] also pointed out as an example that if $\mathsf{succ}(\mathcal{P}) = 0.99$ then to achieve $\mathsf{p} = 0.95$ requires $t = 300$, and to achieve $\mathsf{p} = 0.99$ requires $t = 460$. They further comment that the required number of samples, t, is in fact independent of γ, since it is based instead on the required threshold for $\mathsf{succ}(\mathcal{P})$. This observation applies equally to our analysis.

Dodis, Vadhan, and Wichs [33] used an approach similar to Ateniese *et al.* [3]. They also proposed the use of a *hitting sampler* that amounts to choosing which t elements to sample from a specified distribution that contains fewer than $\binom{n}{t}$ possible sample sets but still guarantees that $\mathsf{Pr}_{\mathsf{bad}}$ is higher than some specified value for a given value of $\mathsf{succ}(\mathcal{P})$ that is less than 1.

These two analyses can be interpreted in the context of a hypothesis test. More precisely, we wish to distinguish the null hypothesis

$\mathsf{Hyp}_0 :\ \mathsf{succ}(\mathcal{P}) \leq 0.99;$

from the alternative hypothesis

$\mathsf{Hyp}_1 :\ \mathsf{succ}(\mathcal{P}) > 0.99.$

If 300 challenges are made and all the responses are correct, then a 95% confidence interval for $\mathsf{succ}(\mathcal{P})$ is $(0.99006, 1]$, so there is enough evidence to reject the null hypothesis at the 5% significance level. However, a 99% confidence interval for $\mathsf{succ}(\mathcal{P})$ is $(0.977, 1]$, so there is insufficient evidence to reject the null hypothesis at the 1% significance level. If, on the other hand, 460 challenges were made and all the responses were correct then a 99% confidence interval for $\mathsf{succ}(\mathcal{P})$ is $(0.99003, 1]$ and so in this case there *is* enough evidence to reject the null hypothesis at the 1% significance level.

We note that this is a special case of our analysis. Specifically, Ateniese *et al.* [3] focused on determining the smallest number of challenges for which an entirely correct response constitutes sufficient evidence to reject the null hypothesis at the desired significance level. On the positive side, this is the smallest number of challenges for which it is possible for a prover to assure the user that it has the file. On the negative side, even a single incorrect response results in failure to reject the null hypothesis, regardless of whether it is true. This is where the oversampling of the random challenges is helpful. Taking a larger sample size has the benefit of increasing the probability that a false null hypothesis is rejected, without adversely affecting the probability that a true Hyp_0 fails to be rejected. For example, from Table 4.2 we see that if 90 correct responses out of 100 are observed then there is insufficient evidence to reject the hypothesis $\mathsf{succ}(\mathcal{P}) \leq 0.8$ at the 5% significance level. However, if 180 correct responses out of 200 are observed then there *is* sufficient evidence to reject this null hypothesis at the 5% significance level (in fact we even have enough evidence to reject Hyp_0 at the 1% significance level).

4.6 A Lower Bound on Storage and Communication Requirements

In this section, we prove a bound that applies to *keyed* PoR schemes. Suppose that M is a random variable corresponding to an unencoded file m chosen uniformly at random. Let V be a random variable denoting the information stored by the Verifier (i.e., the key), and let R be a random variable corresponding to the computations performed by an extractor. The uncertainty in computing the correct file is $\mathbf{H}(\mathsf{M}|\mathsf{V};\mathsf{R})$. Now, from basic entropy inequalities (see Chapter 2), we have

$$\begin{aligned}\mathbf{H}(\mathsf{M}|\mathsf{V};\mathsf{R}) &= \mathbf{H}(\mathsf{M};\mathsf{V};\mathsf{R}) - \mathbf{H}(\mathsf{V};\mathsf{R})\\ &\geq \mathbf{H}(\mathsf{M};\mathsf{V};\mathsf{R}) - \mathbf{H}(\mathsf{V}) - \mathbf{H}(\mathsf{R})\\ &\geq \mathbf{H}(\mathsf{M}) - \mathbf{H}(\mathsf{V}) - \mathbf{H}(\mathsf{R}).\end{aligned}$$

Suppose that the file can be reconstructed by the extractor with probability 1. Then we have $\mathbf{H}(\mathsf{M}|\mathsf{V};\mathsf{R}) = 0$. The inequality proven above imples that

$$\mathbf{H}(\mathsf{M}) \leq \mathbf{H}(\mathsf{V}) + \mathbf{H}(\mathsf{R}). \tag{4.17}$$

Now suppose that the extractor is a black-box extractor. In this situation, we have that

$$\mathbf{H}(\mathsf{R}) \leq \gamma \log_2 |\Delta|, \tag{4.18}$$

since there are γ possible challenges and each response is from the set Δ. The file m is a random vector in $(\mathbb{F}_q)^k$, so

$$\mathbf{H}(\mathsf{M}) = k \log_2 q. \tag{4.19}$$

Therefore, combining (4.17), (4.18) and (4.19), we have the following result.

Theorem 4.15. *Suppose we have a keyed* PoR *scheme where the file is a uniformly random vector in* $(\mathbb{F}_q)^k$*, there are* γ *possible challenges and each response is from the set* Δ*. Suppose that a black-box extractor succeeds with probability equal to* 1*. Then the entropy of the verifier's storage, denoted* $\mathbf{H}(\mathsf{V})$*, satisfies the inequality*

$$\mathbf{H}(\mathsf{V}) \geq k \log_2 q - \gamma \log_2 |\Delta|.$$

Naor and Rothblum [62] proved a lower bound for a weaker form of PoR-type protocol, called an "authenticator". As noted in Dodis, Vadhan, and Wichs [33], the Naor and Rothblum bound also applies to PoR schemes. Phrased in terms of entropy, their bound states that

$$\mathbf{H}(\mathsf{M}) \leq \mathbf{H}(\mathsf{V}) \times \mathbf{H}(\mathsf{R}),$$

which is a weaker bound than (4.17).

In the case of an unkeyed scheme, the extractor is only given access to the proving algorithm. Therefore, $\mathbf{H}(\mathsf{M}|\mathsf{R}) = 0$ if a black-box extractor succeeds with probability equal to 1. From this, it follows that $\mathbf{H}(\mathsf{M}) \leq \mathbf{H}(\mathsf{R})$ in this situation.

4.7 Conclusion

There have been great advances in secure cloud storage over the last decade. Most of these constructions are secure in the computational setting. In all the efficient constructions of PoR systems in the computational setting, the success of extraction depends on the distance of the code used by the client to encode the file before storing it on the server. In these schemes, the distance of the underlying code plays a very pivotal role; the distance of this code is (roughly) inversely proportional to the success probability of the server required to guarantee successful extraction.

We have performed a comprehensive analysis of the extraction properties of unconditionally secure PoR schemes, and established a methodology that is applicable to the analysis of further new schemes. What constitutes "good" parameters for such a scheme depends on the precise application, but our framework allows a flexible trade-off between parameters.

Chapter 5

Extraction or Possession?

The two widely accepted definitions that capture the integrity of cloud storage are both modelled in the terms of the "extractor" paradigm. In the case of PoR systems, extraction is treated as an algorithm that uses the proving algorithm to output the file. On the other hand, in the case of PDP systems, the definition uses the knowledge extractor of Bellare and Goldreich [9]. This similarity results in Question 2 raised in Chapter 1.

In this chapter, we answer Question 2. Recall that in a PoR scheme [51], a theorem stating the security of the system has the following form: there exists an Extractor with the property that it can recover M whenever $\mathsf{succ}(\mathcal{P})$ is sufficiently high for the given proving algorithm $\mathcal{P}$. A PoR scheme is quantified by the efficiency of the extractor and two parameters: $0 < \nu \leq 1$ and $0 < \eta \leq 1$. We say that a PoR system is an (η, ν)-PoR system provided that there exists an Extractor whose success probability is at least ν whenever $\mathsf{succ}(\mathcal{P}) \geq \eta$. Recall that $\mathsf{succ}(\mathcal{P})$ is the probability that $\mathcal{P}$ gives the correct response to a random challenge; the success probability of an Extractor is the probability that it is able to extract the entire file M correctly. The success probability of an Extractor takes into account any random choices it makes during its execution.

On the other hand, the definition of a PDP system as appeared in Ateniese *et al.* [4] and mentioned in Chapter 3 also uses an Extractor but in a slightly different manner. More concretely, a PDP system is said to be secure if the success probability of an adversary and the knowledge extractor of Bellare and Goldreich [9] are negligibly close. However, there are few issues with the direct use of the extractor of Bellare and Goldreich [9] which we enumerated in Chapter 3.

In the rest of this chapter, we study this issue in more detail. The content of this chapter is based on Stinson and Upadhyay [80]:

Douglas R. Stinson, Jalaj Upadhyay. Is extracting data the same as possessing data? J. Mathematical Cryptology 8(2): 189-207 (2014).

5.1 Defining Possession of Data

It is often stated in the literature that PoR is a stronger requirement than PDP systems (see, for example, the related work section of [33, 72]). Our main goal is to separate the notions of "retrievability" and "possession". To make a distinction between retrievability and possession, it is helpful to ask what it means to actually "possess" data. We use the approach of Goldreich [40] to define PoK systems, and concentrate on the behavioural meaning of the term "possession". We might argue that possessing data means that the server has an exact copy of the file M in its storage at a given time. But it is not clear how can one force the server to store the exact copy of the file. In fact, it might be even impossible to achieve this against a malicious server because the server can store M in some altered form, for example, by using some kind of compression. Even with an honest server, it might be possible that the server stores the file in an encrypted form to protect against unwanted security loss. However, it is conceivable that the challenge-response protocol might force the reconstruction of M from whatever form the server has the file, in the sense that a response cannot be generated without first restoring M to its initial form. In this case, an exact copy of M would exist in the server's memory space at the time the response to the challenge is computed, and we could say that the server "possesses" the data at this point in time.

Based on the above discussion, we now define a *proof-of-data-observability scheme* (PDO *scheme*). This new definition captures the idea that the server has the whole file M during the execution of $\mathcal{P}$. To formalize this stronger security notion, we consider an *observer* $\mathcal{O}$ that has considerably less power than an extractor.

Definition 5.1. *An observer $\mathcal{O}$ is an entity that is given read access to the code describing the proving algorithm $\mathcal{P}$. It is permitted to do the following:*

- *$\mathcal{O}$ is allowed to observe a single run of the proving algorithm $\mathcal{P}$ with a random challenge supplied by the client,*
- *$\mathcal{O}$ can perform whatever additional computations it likes, and*
- *$\mathcal{O}$ should copy the correct file M from the contents of $\mathcal{P}$'s memory space to an output tape, at some time during the execution of $\mathcal{P}$.*

Let us look closely at the above definition. Intuitively, this definition captures the idea that $\mathcal{O}$ is able to take a "snapshot" of the memory of $\mathcal{P}$ at some precise time when it contains the file M. This is in alignment with the discussion above that the file might be stored in some other form, but the challenge-response protocol cannot be carried out without restoring the file to its proper form.

The term "copy" is very crucial here. For example, we restrict the observer from performing any post-processing once it copies the file. Moreover, we do not place any restriction on the proving algorithm. For example, we allow the situation where the proving algorithm has access to the file in some other form, like a compressed form, and reconstructs the whole file online before it responds to the query.

We are now ready to define a PDO system.

Definition 5.2. *Let $0 < \nu \leq 1$ and let $0 < \eta \leq 1$. We say that an observer $\mathcal{O}$ is an (η, ν)-*observer *provided that the expected fraction of M that $\mathcal{O}$ copies is at least ν whenever $\mathsf{succ}(\mathcal{P}) \geq \eta$. A system for which there exists an (η, ν)-observer is called a* proof-of-data-observablity *system.*

Compared to a PoR scheme, for a PDO scheme, we have a somewhat different objective. The above definition captures this difference. More precisely, it captures the notion that, on average, $\mathcal{O}$ is able to copy at least $\nu|M|$ bits of M, where $|M|$ is the bit-length of the file. This average is computed over the random challenges chosen by the client, as well as any random choices the observer makes during its execution.

The above definition has several subtle aspects that distinguishes it from PoR and PDP systems. In a PDO system, an observer is allowed to see only a single run of the challenge-response protocol that is being executed with a random challenge chosen by the user. This is in contrast with the PoR system where the extractor has access to the proving algorithm and can make queries on behalf of the user. If the observer is not allowed to know the code of the proving algorithm, there is no way such a scheme can be realized. To an extent, it is the observer's knowledge of how the proving algorithm works that allows it to recognize the presence of the file M in the memory space of $\mathcal{P}$ as it is executed, and copy it to its output tape.

Remark 3. We also mention that there is an intrinsic limitation of PDO systems. To see this, consider an n-block file M. Further assume that a response in a challenge-response protocol depends on a random subset of n' message blocks chosen by the client. For $\eta > 0$, the best observer one could hope for would be a $(\eta, n'/n)$-observer, because a single run of $\mathcal{P}$ only requires performing a computation on $n'|M|/n$ bits of the file M, where $|M|$ is the bit length of the file M. On the other hand, a trivial example of a PDO scheme that has a $(1,1)$-observer would be one in which the server must give the entire file M to the client all at once. □

5.1.1 PDO is a Stronger Notion than PoR and PDP

The definition of PDO is more stringent in comparison with PoR or PDP systems in the sense that the observer is allowed only a single run of the challenge-response protocol. In this section, we give a simple example that is a PoR (and PDP scheme) but not a PDO scheme.

Let us consider the Linear Combination Scheme. Let $M = (M_1, \ldots, M_n) \in (\mathbb{F}_q)^n$ be the file stored on the remote server and $c = (c_1, \ldots, c_n)$ be a challenge consisting of random entries picked from $\mathbb{F}_q$. The response function is defined as $\varrho = \langle M, c \rangle$. It is a PoR system due to Theorem 4.7 and PDP scheme due to the result of Ateniese *et al.* [3].

On the other hand, we explicitly show an adversary that can generate correct responses to the challenges without explicitly storing or performing computations on the original file M. The adversary works as follows: (i) when the Adversary is originally given the file M, it chooses a random $\gamma \in \mathbb{F}_q$ such that $\gamma \neq 1$, (ii) the Adversary computes $M' = \gamma M$, and (iii) the Adversary stores (M', γ) and discards M. Later, when the adversary receives a challenge c, it computes the (correct) response $r = \gamma^{-1}(c \cdot M')$. This computation does not use M, so no observer can obtain M simply by monitoring the online computations of the adversary. Therefore, this is not a PDO scheme.

5.1.2 Our Contributions

In this chapter, we investigate PDO schemes in the *random oracle model*. We follow the usual random oracle model as defined in Bellare and Rogaway [10], where hash is regarded as a function that takes as input a binary string of arbitrary length and produces a completely random h-bit output. In our scheme, the client precomputes a set of challenge-response pairs for a file M. The challenge is picked randomly from $(\mathbb{F}_2)^\ell$ and the response is of the form $\varrho = \mathsf{hash}(M \parallel c)$ for a challenge c. During an audit, it challenges the server with one of its precomputed challenges. Our proof relies on an important property of random oracles, namely that any hash value $\mathsf{hash}(x)$ is completely independent of all the other hash values $\mathsf{hash}(x')$ for $x' \neq x$. This fact is used either implicitly or explicitly in our analyses of the protocols we discuss.

As in Chapter 4, we do not restrict the computational abilities of the server; i.e., we analyze our schemes in the setting of unconditional security. However, we need some restrictions on the server. For example, a server with unbounded storage can store all the possible responses to ℓ-bit challenges and discard the file. Therefore, we must either bound the amount of storage available to the adversarial servers and/or the number of random oracle calls that a server can make.

As a warm-up, we analyze our hash-based challenge-reponse scheme against an adversary that stores part of the file along with some number of precomputed responses in Section 5.2. In Section 5.3, we consider a more general adversary that is only restricted in the number of accesses to the random oracle that it can make. Both these schemes require the server to go through the whole file. As mentioned in Remark 3, we cannot improve on the guarantee of a PDO scheme if the server does not need to see the whole file in order to respond to a challenge. In practice, this may be little problematic. To remedy these concerns, in Section 5.4, we consider observers who watch $s > 1$ executions of the challenge-response protocol, obtaining part of the file each time the protocol is run.

5.2 A Basic PDO Scheme

We begin by presenting a simple PDO scheme in Figure 5.1 which we prove secure under the random oracle model. The scheme is based on hashing the file along with a random challenge provided by the client. We first give a brief overview of the basic idea behind our proof.

The key idea behind proving that it is a PDO scheme is as follows. As $\mathsf{hash}(\cdot)$ is a random oracle, the only way to compute $\mathsf{hash}(M \parallel c)$ is to supply hash with all the bits in M and c. However, a possible complication is that a response $\mathsf{hash}(M \parallel c)$ could be *precomputed* ahead of time by $\mathcal{P}$, instead of being computed *online* during the execution of the challenge-response protocol $\mathcal{P}$. Now if we upper-bound the number of oracle accesses that are permitted in the pre-computation stage, then we can upper-bound the number of precomputed responses that $\mathcal{P}$ can utilize. If $\mathcal{P}$ is required to have a sufficiently high success probability and c is chosen from a large enough challenge space, then it must be the case that, with high probability, $\mathsf{hash}(M \parallel c)$ will be computed online during $\mathcal{P}$'s execution. In this case, the file M is necessarily stored in $\mathcal{P}$'s memory space immediately prior to the call $\mathsf{hash}(M \parallel c)$ that is made to the random oracle hash[1]. Therefore we will have a PDO scheme, according to our definition given above. We now continue with a formal analysis.

[1] When we call the function $\mathsf{hash}(x)$, we are assuming that the entire input is being specified at that time, e.g., by passing a memory address that contains the input x (e.g., a standard call-by-reference).

Precomputation: The client and the server are given access to a random oracle hash that maps an arbitrary length input to a h-bit output. The client gives a file M to the server, where the length of M is n bits. The server is permitted to access hash some fixed number of times, and then the server creates a proving algorithm $\mathcal{P}$.

Challenge: The client chooses an ℓ-bit random challenge, c, and sends it to the server.

Response: The server computes the response $r = \mathcal{P}(c)$ and sends it to the client[a].

Verification: The client accepts the response r as valid if $r = \mathsf{hash}(M \parallel c)$.

[a] We can assume, without any loss of generality, that the random oracle is accessed at most once in the computation of the response.

Figure 5.1: Basic Random Oracle PDO scheme

As defined earlier, the server constructs the proving algorithm $\mathcal{P}$ after it has been given a copy of the file M. The server is also permitted to access the random oracle and use the results to help construct $\mathcal{P}$. For example, various responses of the form $\mathsf{hash}(M \parallel c)$ could be precomputed and stored by $\mathcal{P}$.

We consider a file that is n bits long and every challenge c is ℓ bits long. In other words, a file is picked from a message space $\mathbb{F}_2^n$ and the challenge space is $\mathbb{F}_2^\ell$. In this scheme, h and ℓ will typically be much smaller than n. As an illustrative example, we could take $\ell = 100$, $h = 256$, and $n = 2^{25}$.

We will be analyzing the scheme in Figure 5.1 in the random oracle model. Nevertheless, it might be of interest to consider how this scheme might be instantiated in practice, where one would replace the random oracle with a "real" hash function. We note that this would have to be done carefully in order to prevent certain types of attacks. For example, if $\mathsf{hash}(\cdot)$ is an iterated hash function, then $\mathsf{hash}(M)$ can be precomputed and stored, and then $\mathsf{hash}(M \parallel c)$ can be computed from $\mathsf{hash}(M)$ and c. These types of issues are studied in detail in Ristenpart, Shacham, and Shrimpton [69]. One good practical solution might be to use a *zipper hash* construction [58].

In this section, we give an exact combinatorial analysis of a specific type of adversary (i.e., a proving algorithm). Our adversary is a *storage-bounded* server with t bits of storage and it uses s bits to store (a fraction of) M. We use the notation $\mathsf{Adv}(t, s)$ to denote such an adversary. This storage will be used to store part or all of the file M and/or some number of precomputed responses. Although not completely general, this type of adversary is a very natural and plausible one to consider.

It may be useful to distinguish adversaries according to their intent. Perhaps the adversary is just being *selfish* in that it wants to achieve a "reasonable" success probability without actually storing the whole file. The adversary $\mathsf{Adv}(t, s)$ would be selfish only if $t < n$. Alternatively, the adversary could be *malicious*, which means that he might be willing to devote a large amount of storage to precomputed responses in an attempt to deceive the client who is expecting the server to store the file. The adversary $\mathsf{Adv}(t, s)$ would be malicious only if $t \geq n$.

The adversary $\mathsf{Adv}(t, s)$ stores s bits of M, along with s^* precomputed responses obtained from s^* accesses to the random oracle. Clearly, $s \leq \min\{n, t\}$. Since each response has h bits, it must be the case that

$$s^* \leq \left\lfloor \frac{t-s}{h} \right\rfloor .$$

Additionally, there are 2^ℓ possible responses, so $s^* \leq 2^\ell$. Therefore, for the adversary $\mathsf{Adv}(t, s)$, we have

$$s^* \leq \min \left\{ 2^\ell, \left\lfloor \frac{t-s}{h} \right\rfloor \right\} .$$

The data precomputed by $\mathsf{Adv}(t, s)$ can be thought of as a list of ordered pairs of the form $(c, \mathsf{hash}(M \parallel c))$. In our analysis, we are only counting the storage needed for the responses $\mathsf{hash}(M \parallel c)$, as various methods could be used to reduce or even eliminate the storage required for the challenges (i.e., the c values). For example, the stored responses might correspond to s^* consecutive values of c.

Given a challenge c, $\mathsf{Adv}(t, s)$ operates as follows:

1. If the (correct) response for a challenge c has been precomputed, say $\varrho = \mathsf{hash}(M \parallel c)$, then $\mathcal{P}$ outputs ϱ.
2. If the $\mathsf{Adv}(t, s)$ receives a challenge c for which it has not been precomputed and stored the response, then $\mathcal{P}$ randomly guesses the $n - s$ non-stored bits of M, creating a file M^*. The adversary then computes $\varrho = \mathsf{hash}(M^* \parallel c)$ and outputs r.

Note that the adversary $\mathsf{Adv}(t, s)$ makes at most one (on-line) call to the random oracle each time the challenge-response protocol is run.

We do a case-by-case analysis. The case where $t \geq 2^\ell h$ is not very interesting because the adversary can precompute all 2^ℓ possible responses when $s = 0$. Therefore, the resulting proving algorithm has success probability equal to 1 and no bits of the file M are stored. Therefore, in what follows, we assume that $t < 2^\ell h$, which implies that

$$s^* = \left\lfloor \frac{t-s}{h} \right\rfloor. \tag{5.1}$$

Theorem 5.1. *Let t, ℓ, s, and h be as defined above. Assume that $t < 2^\ell h$. Then the success probability of $\mathsf{Adv}(t, s)$ is*

$$\mathsf{p}(t,s) = 1 - \left(1 - \frac{2^s}{2^n}\right)\left(1 - \frac{1}{2^h}\right)\left(1 - \frac{t-s}{2^\ell h}\right). \tag{5.2}$$

Proof. For the specific space-bounded adversary defined above, we have three different events to consider: (i) when the adversary has precomputed the response, (ii) when the adversary correctly guess the remaining $n - s$ bits of the file, and (iii) when the random oracle outputs the correct response even if the input is incorrect. Note that in each of these three events, the adversary manages to convince the user that it has the file even though it does not have the file. For the more formal proof, we bound the probability of each of these events.

Let E_1 denote the event that $\mathsf{Adv}(t, s)$ has precomputed the response to a given random challenge. Let E_2 denote the event that $\mathsf{Adv}(t, s)$ correctly guesses the $n - s$ non-stored bits of n. Finally, let E_3 denote the event that the on-line output of the random oracle on an incorrect input yields the correct response (i.e., $\mathsf{hash}(x) = \mathsf{hash}(M \parallel c)$ for a given $x \neq M \parallel c$). For $1 \leq i \leq 3$, let $\mathsf{p}_i = \mathsf{Pr}[E_i]$. From equation (5.1), we have

$$\mathsf{p}_1 = \frac{s^*}{2^\ell} = \frac{t-s}{2^\ell h}.$$

Here, we are assuming for convenience that $(t - s)/h$ is an integer, so we can omit the "floor" computation. It is also clear that

$$\mathsf{p}_2 = \frac{2^s}{2^n} \quad \text{and} \quad \mathsf{p}_3 = \frac{1}{2^h}.$$

Now the success probability of $\mathsf{Adv}(t, s)$ is

$$\begin{aligned} \mathsf{p}(t,s) &= \mathsf{p}_1 + (1-\mathsf{p}_1)(\mathsf{p}_2 + (1-\mathsf{p}_2)\mathsf{p}_3) \\ &= 1 - (1-\mathsf{p}_1)(1-\mathsf{p}_2)(1-\mathsf{p}_3), \end{aligned} \tag{5.3}$$

which simplifies to give the desired result. □

Suppose we fix a value of $t < 2^\ell h$ and, for brevity, we denote $\mathsf{p}(s) = \mathsf{p}(t, s)$. We will analyze the behaviour of $\mathsf{p}(s)$ as a function of s. In order to do this, it is helpful to define

$$\mathsf{q}(s) = (2^\ell h - t + s)(2^s - 2^n).$$

It is easy to see that

$$\mathsf{q}(s) = C(\mathsf{p}(s) - 1), \tag{5.4}$$

where C is a constant (independent of s). Using elementary calculus, we have

$$\mathsf{q}'(s) = 2^s(1 + (2^\ell h - t + s)\ln 2) - 2^n. \tag{5.5}$$

Taking the second derivative of $\mathsf{q}(s)$, from Equation (5.5) it is easy to see that $\mathsf{q}''(s) > 0$ for all $s > 0$. This in combination with Equation (5.4) implies that $\mathsf{p}''(s) > 0$ for all $s > 0$, so the function $\mathsf{p}(s)$ is "concave up".

First, let us consider the situation where $s = 0$. We have

$$\mathsf{q}'(0) = (1 + (2^\ell h - t)\ln 2) - 2^n.$$

Under the reasonable assumption that $2^\ell h \lll 2^n$, we will have $\mathsf{q}'(0) < 0$ and $p'(0) < 0$.

At this point, we have shown that $p'(0) < 0$ and $p'(s)$ is an increasing function of s. The maximum value of $\mathsf{p}(s)$ must occur at one of the two endpoints (i.e., when $s = 0$ or when $s = \min\{n, t\}$). Therefore, there are two possibilities to consider:

case (1) The maximum value of $\mathsf{p}(s)$ is attained when $s = 0$.

case (2) The maximum value of $\mathsf{p}(s)$ is attained when $s = \min\{n, t\}$. Note that this case can happen only when $\mathsf{p}'(\min\{n, t\}) > 0$.

In order to determine which case would apply to a given parameter situation, it is helpful to estimate the value $\tilde{s} > 0$ such that $\mathsf{p}(0) = \mathsf{p}(\tilde{s})$. From (5.2), we see that $\mathsf{p}(0) = \mathsf{p}(\tilde{s})$ if and only if

$$\left(1 - \frac{1}{2^n}\right)\left(1 - \frac{t}{2^\ell h}\right) = \left(1 - \frac{2^{\tilde{s}}}{2^n}\right)\left(1 - \frac{t - \tilde{s}}{2^\ell h}\right).$$

For large n and ℓ, we can ignore terms that are products of two "small" numbers. Ignoring these terms, the equation to be solved for $\tilde{s}$ can be approximated as

$$\frac{t}{2^\ell h} + \frac{1}{2^n} \approx \frac{t - \tilde{s}}{2^\ell h} + \frac{2^{\tilde{s}}}{2^n}.$$

For $\tilde{s}$ reasonably large, we can ignore the term $1/2^n$. In this case, we can further simplify the above equation as

$$\frac{2^{\tilde{s}}}{2^n} \approx \frac{\tilde{s}}{2^\ell h},$$

which is equivalent to

$$\tilde{s} + \ell + \log_2 h \approx \log_2 \tilde{s} + n.$$

For $\tilde{s}$ reasonably large, we can ignore the term $\log_2 \tilde{s}$. Thus we finally obtain the estimate $\tilde{s} \approx s'$, where

$$s' = n - \ell - \log_2 h. \tag{5.6}$$

We see that $s' > 0$ because we assumed that $2^\ell h < 2^n$. Now, in order for **case (2)** to occur, we need $s' \leq \min\{n, t\}$. It is clear that $s' < n$. However, it may or may not be the case that $s' \leq t$.

The two possible cases mentioned above will be distinguished by whether $s' \leq t$ or $s' > t$. Roughly speaking, we have the following possibilities:

1. If $t < s'$, then $\mathsf{p}(s) \leq \mathsf{p}(0)$ for $0 < s < \min\{n, t\}$.
2. If $t \geq s'$, then $\mathsf{p}(s) > \mathsf{p}(0)$ for $s' < s < \min\{n, t\}$.

The second case is of course the more interesting one. This is because, in the first case, the adversary is better off storing nothing.

The above discussion can be rephrased by saying that, if the adversary's success probability exceeds $\mathsf{p}(0)$, then $s' \leq s \leq t$. In this situation, the adversary must be storing all of the file M, except for $\ell + \log_2 h$ (or fewer) bits.

We present some numerical examples.

Example 15. Suppose $n = 2^{12} = 4096$, $\ell = 50$ and $h = 64$. Then $s' = 4040$. The exact value for $\tilde{s}$ is $\tilde{s} = 4041$, so the estimated value s' from (5.6) is very accurate. Suppose the adversary's storage is $t = s'$ bits (this would be a selfish adversary). Then $\mathsf{p}(0) = 5.6 \times 10^{-14}$. Therefore, if the adversary's success probability exceeds 5.6×10^{-14}, it must be the case that the adversary is storing at least 4041 of the 4096 bits of M (i.e., all but 55 bits). Figure 5.2 presents a graph of $\mathsf{p}(s)$ as a function of s. Note that $\mathsf{p}(s)$ is very close to 0 until s is almost n, and then $\mathsf{p}(s)$ increases very quickly to 1. This behaviour is typical of most "reasonable" parameter situations.

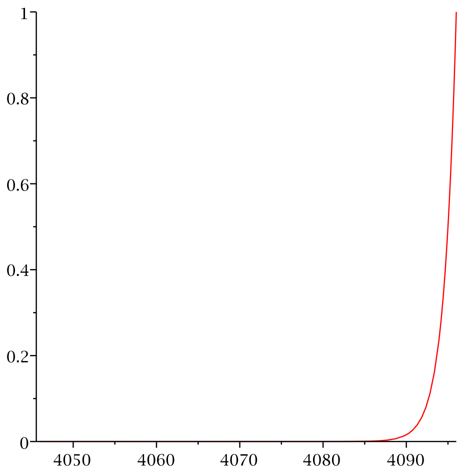

Figure 5.2: Success Probability as a Function of s when $n = 4096$, $\ell = 50$ and $h = 64$

Example 16. Suppose $n = 2^{25} = 33554432$, $\ell = 100$ and $h = 256$. Then $s' = 33554324$. The exact value for $\tilde{s}$ is $\tilde{s} = 33554329$, so the estimated value s' from (5.6) is very accurate. Suppose the (selfish) adversary's storage is $t = s'$ bits. Then $\mathsf{p}(0) = 1.03 \times 10^{-25}$. Therefore, if the adversary's success probability exceeds 1.03×10^{-25}, it must be the case that the adversary is storing at least 33554329 of the 33554432 bits of M (i.e., all but 103 bits).

Now we consider how to construct an observer $\mathcal{O}(t, s)$ for the adversary $\mathsf{Adv}(t, s)$ and we analyze the success probability of the observer we construct. $\mathcal{O}(t, s)$ will simply copy any value M where the output of $\mathcal{P}$ is obtained by an online evaluation of $\mathsf{hash}(M \parallel c)$, where c is the challenge. Suppose we define events E_i and probabilities p_i $(1 \leq i \leq 3)$ as in the proof of Theorem 5.1. It is clear that $\mathcal{O}(s, t)$ outputs the correct file M whenever E_2 holds and E_1 does not hold. Therefore the success probability of $\mathcal{O}(s, t)$ is

$$(1 - \mathsf{p}_1)\mathsf{p}_2 = \left(1 - \frac{t - s}{2^\ell h}\right) 2^{s-n}.$$

If the success probability of $\mathcal{O}(s, t)$ is denoted by ν, then, from (5.3), we can compute

$$\nu = \mathsf{p}(t, s) - \mathsf{p}_1 - (1 - \mathsf{p}_1)(1 - \mathsf{p}_2)\mathsf{p}_3.$$

It is clear that

$$\nu > \mathsf{p}(t, s) - \mathsf{p}_1 - \mathsf{p}_3. \tag{5.7}$$

For most "reasonable" parameter values, p_1 and p_3 are very small, and hence ν is very close to $\mathsf{p}(s, t)$.

A Variation of the Adversarial Strategy. One possible variation/generalization of the adversary $\mathsf{Adv}(t, s)$ would be to consider storing *partial* precomputed information about responses rather than "all or nothing". However, it is not hard to see that this will not increase the adversary's success probability. Suppose that an adversary $\mathsf{Adv}(t, s)$ stores h_1 bits of the response $\mathsf{hash}(M \parallel c_1)$ and h_2 bits of the response $\mathsf{hash}(M \parallel c_2)$, where $0 < h_1 \leq h_2 < h$. Consider the modified adversary $\mathsf{Adv}'(t, s)$ who stores $h_1 - 1$ bits of the response $\mathsf{hash}(M \parallel c_1)$ and $h_2 + 1$ bits of the response $\mathsf{hash}(M \parallel c_2)$. These two adversaries have the same storage requirement. However, we show below that the success probability of $\mathsf{Adv}'(t, s)$ is at least as large as the success probability of $\mathsf{Adv}(t, s)$. Iterating this process enough times, we can transform the adversary $\mathsf{Adv}(t, s)$ into one that does not store any "partial" responses.

Here is the proof of the claim: The probability of Adv guessing the response is 2^{-h+h_1} when the challenge is c_1, and 2^{-h+h_2} when the challenge is c_2. For the adversary Adv', the probabilities are 2^{-h+h_1-1} when the challenge is c_1, and 2^{-h+h_2+1} when the challenge is c_2. The claim is proven by observing that

$$2^{-h+h_1} + 2^{-h+h_2} \leq 2^{-h+h_1-1} + 2^{-h+h_2+1}$$

when $h_1 \leq h_2$.

5.2.1 Discussion

We conclude this section by discussing how the results presented in this section should be interpreted. Theorem 5.1 gives an exact success probability of a particular type of bounded storage adversary in terms of different parameters. If parameters of the scheme are chosen in a suitable way, this probability is close to 1 only when $\tilde{s}$, the number of bits of the file stored properly by the adversary, is very close to the actual size of the file. Therefore, if the adversary succeeds in the challenge-response protocol with a probability close to 1, then this implies that it has stored almost all of the file. The construction of the observer and Equation (5.7) then guarantee that the observer outputs almost all of the file with probability very close to the success probability of the adversary.

5.3 A General Bound

In the last section, we considered a specific class of adversary that has bounded storage. In this section, we consider a much more general adversary. However, we still need some restriction on the adversary; otherwise, the adversary can simply compute all possible responses. Instead of bounding the storage of an arbitrary proving algorithm like we did in the last section, we bound the number of calls the adversary can make to the random oracle during the pre-computation phase and before defining the proving algorithm. We denote the bound on the number of random oracle calls permitted by s^*. By the end of the pre-computation phase, the adversary constructs a proving algorithm $\mathcal{P}$. We assume the following about the proving algorithm $\mathcal{P}$:

1. The proving algorithm $\mathcal{P}$ is an arbitrary algorithm for which $\mathsf{succ}(\mathcal{P}) \geq \eta$.

2. We allow $\mathcal{P}$ to store all the responses to the random oracle made during the pre-computation phase.

We refer to the resulting proving algorithm $\mathcal{P}$ as a (η, s^*)-*adversary.* $\mathcal{P}$ can be either a deterministic or a randomized algorithm.

Since there are at most 2^ℓ possible challenges, we can assume without loss of generality that $s^* \leq 2^\ell$.

First, we define

$$S_P := \{c : \mathsf{hash}(c \parallel M) \text{ has been precomputed by } \mathcal{P}\}.$$

Observe that $|S_P| \leq s^*$.

Second, for $c \notin S_P$, let p_c denote the probability that $\mathcal{P}$ computes $\mathsf{hash}(c \parallel M)$ online, in response to the challenge c.

Theorem 5.2. *Let the challenge space be* $\{0,1\}^\ell$ *and let* h *be the size of the output of* hash. *Suppose that* $\mathcal{P}$ *is a* (η, s^*)*-adversary for a challenge-response protocol. Then*

$$2^\ell \eta \leq \left(1 - \frac{1}{2^h}\right)\left(s^* + \sum_{c \notin S_P} \mathsf{p}_c\right) + 2^{\ell-h}. \tag{5.8}$$

Proof. There are $|S_P|$ values of c for which $\mathsf{hash}(c \parallel M)$ has been precomputed. For a challenge $c \notin S_P$, $\mathsf{hash}(c \parallel M)$ is computed online with probability p_c. If $c \notin S_P$, then with probability $1 - \mathsf{p}_c$, the response $\mathsf{hash}(c \parallel M)$ is not computed online. In this situation the probability that the response given by $\mathcal{P}$ is correct is 2^{-h}. It follows that

$$\mathsf{succ}(\mathcal{P}) \leq \frac{|S_P|}{2^\ell} + \frac{1}{2^\ell} \sum_{c \notin S_P} \left(\mathsf{p}_c + (1 - \mathsf{p}_c)2^{-h}\right).$$

After some simplification, the right-hand side of the above inequality can be rewritten as

$$\mathsf{succ}(\mathcal{P}) \leq \frac{1}{2^\ell}\left(1 - \frac{1}{2^h}\right)\left(|S_P| + \sum_{c \notin S_P} \mathsf{p}_c\right) + 2^{-h}.$$

Using the fact that $|S_P| \leq s^*$, we obtain

$$\mathsf{succ}(\mathcal{P}) \leq \frac{1}{2^\ell}\left(1 - \frac{1}{2^h}\right)\left(s^* + \sum_{c \notin S_P} \mathsf{p}_c\right) + 2^{-h}.$$

Since $\mathsf{succ}(\mathcal{P}) \geq \eta$, the inequality (5.8) follows. □

Next, we consider how to construct an observer $\mathcal{O}$ for $\mathcal{P}$. The basic idea is to copy any value M such that the output of $\mathcal{P}$ is obtained by an online evaluation of $\mathsf{hash}(M \parallel c)$, where c is the challenge. However, a possible complication would arise if $\mathsf{hash}(M \parallel c) = \mathsf{hash}(M' \parallel c)$ for some $M' \neq M$ and for some c. For example, perhaps $\mathcal{P}$ computes $\mathsf{hash}(M \parallel c)$, and then it manages to find $M' \neq M$ such that $\mathsf{hash}(M \parallel c) = \mathsf{hash}(M' \parallel c)$. It might not be obvious to the observer whether to copy M or M'.

One possible way to address this issue would be to assume that the random oracle is *collision resistant*. In this case, even though collisions for hash exist, the algorithm $\mathcal{P}$ is not able to find any of these collisions. Such an assumption would be plausible if $h > 160$, say, since the complexity of a collision-finding algorithm for a random oracle with an h-bit hash output is roughly $\Theta(2^{h/2})$ (e.g., as a consequence of the birthday paradox [79]).

Theorem 5.3. *Let ℓ and h be as defined in Theorem 5.2. Suppose that $\mathcal{P}$ is an (η, s^*)-adversary for a challenge-response protocol, and suppose that the random oracle* hash *is collision-resistant. Then there exists an (η, ν)-observer for $\mathcal{P}$, where*

$$\nu \geq \frac{2^h \eta - 1}{2^h - 1} - \frac{s^*}{2^\ell}.$$

Proof. As described above, the observer $\mathcal{O}$ for $\mathcal{P}$ copies any value M such that the output of $\mathcal{P}$ is obtained by an online evaluation of $\mathsf{hash}(M \parallel c)$, where c is the challenge. Denote

$$\mathsf{p}^* = \sum_{c \notin S_P} \mathsf{p}_c.$$

Since the random oracle is collision-resistant, it follows that $\mathcal{O}$ is an (η, ν)-observer, where $\nu = \mathsf{p}^*/2^\ell$. Appyling (5.8) from Theorem 5.2 and simplifying, it follows that

$$\nu = \frac{\mathsf{p}^*}{2^\ell} \geq \frac{2^h\eta - 1}{2^h - 1} - \frac{s^*}{2^\ell}.$$

□

Let us look a bit more closely at the lower bound for ν proven in Theorem 5.3, namely,

$$\frac{2^h\eta - 1}{2^h - 1} - \frac{s^*}{2^\ell}.$$

For reasonable values of the parameters, the first term is essentially equal to η and the second term can be ignored. As an example, suppose we take $\ell = 100$ and $h = 256$. Even if the adversary precomputes $s^* = 2^{60}$ responses, the second term is at most 2^{-40}. When η is somewhat larger than 2^{-40}, we see that the lower bound on ν is very close to η.

5.4 Multiple Runs of the Challenge-Response Protocol

In the last two sections, we considered a hash-based scheme secure under the security definition of a PDO system. Our definition of an observer given in Section 5.1 requires that a ν fraction of the file M can be obtained from a *single run* of the challenge-response protocol. On the positive side, security under this definition gives us a stronger guarantee about the file stored on the remote server. On the negative side, any protocol that satisfies this definition must necessarily require a response to a challenge to depend on at least an ν fraction of the file. This can be problematic from the efficiency point of view, especially when ν is close to 1. This is also against one of the motivations of a PoR system — a single response should depend on a fraction of the file and yet should give some integrity guarantee about the entire file through a series of challenges.

In this section, we give a relaxation of our definition of an observer which allows a prover to respond to the challenges more efficiently while preserving the same type of security guarantees considered in Section 5.2 and Section 5.3. Specifically, we allow the observer to watch some number s of executions of the challenge-response protocol, copying part of M from each run. After s runs of the protocol, the observer should have copied a ν fraction of file M, on average.

Definition 5.3. *Let $0 < \nu \leq 1$, let $0 < \eta \leq 1$ and let $s \geq 1$ be an integer. We say that an observer $\mathcal{O}$ is an (η, ν, s)-*observer *provided that the expected fraction of the file M that $\mathcal{O}$ copies is at least ν whenever* $\mathsf{succ}(\mathcal{P}) \geq \eta$*, where the observer watches s runs of the challenge-response protocol with random challenges chosen by the client.*

In general, ν will increase as s is increased.

We give an analysis of the Multiblock Challenge Scheme in this setting. Recall that the file consists of n blocks. Suppose that a challenge c consists of a randomly-chosen k-subset of $\{1, \ldots, n\}$, say $c = \{i_1, \ldots, i_k\}$, where $i_1 < \ldots < i_k$. The correct response is $r = (M_{i_1}, \ldots, M_{i_k})$.

Theorem 5.4. *Let n and k be as defined above. For any integer $s \geq 1$, there is a (η, ν_s, s)-observer for the Multiblock Challenge Scheme, where*

$$\nu_s = \eta \left(1 - \left(1 - \frac{k}{n}\right)^s\right).$$

Proof. Denote s random challenges by $c^1, \ldots, c^s$. The observer $\mathcal{O}$ will simply copy every message block supplied by $\mathcal{P}$. It is possible that $\mathcal{O}$ observes a message block M_i and later observes a possibly different version of the same message block M_i, because some of the adversary's responses might be erroneous. In this case, only the "last" copy of M_i is retained. However, any specific copy of M_i obtained by $\mathcal{O}$ is correct with probability at least η.

For $1 \leq j \leq n$, define a random variable $\mathbf{X}_j$, where

$$\mathbf{X}_j = \begin{cases} 1 & \text{if } j \in \bigcup_{i=1}^{s} c^i \\ 0 & \text{otherwise.} \end{cases}$$

Then define $\mathbf{X} = \sum_{j=1}^{m} \mathbf{X}_j$. It is easy to see that

$$\mathbb{E}[\mathbf{X}_j] = \mathsf{Pr}[\mathbf{X}_j = 1] = 1 - \left(1 - \frac{k}{n}\right)^s$$

for $1 \leq j \leq n$. By linearity of expectation, we have

$$\mathbb{E}[\mathbf{X}] = n\left(1 - \left(1 - \frac{k}{m}\right)^s\right).$$

This says that the expected fraction of message blocks that are queried is $1 - \left(1 - \frac{k}{n}\right)^s$.

The probability that a specific response to a query is correct is at least η. Therefore, our observer is a (η, ν_s, s)-observer, where $\nu_s = \eta\left(1 - \left(1 - \frac{k}{n}\right)^s\right)$. □

For fixed k and m, we have that $\left(1 - \frac{k}{n}\right)^s \approx e^{-ks/n}$, so ν_s approaches η exponentially quickly as a function of s.

5.5 Summary and Conclusion

We have introduced the notion of proof-of-data-observability as a strengthening of the now-standard techniques of proof-of-recovery and proof-of-data-possession schemes for cloud storage. This new concept provides stronger guarantees relating to the server's behaviour than the traditional notions do. We described and analyzed some simple schemes for proof-of-data-observability in the random oracle model. We also proved some general necessary conditions for the existence of these schemes, and we studied an extension where the observer is allowed to accumulate the file over time, by observing a sequence of runs of the challenge-response protocol.

Chapter 6

Multi-server PoR Systems

In the last two chapters, we discussed proof-of-storage systems when only one storage server was involved. However, in the real world, it is highly likely that a client would store its data on more than one storage server. This is due to a variety of reasons. For example, a client might wish to have a certain degree of redundancy if one or more servers fails. In this case, the client is more likely to store multiple copies of the same data. Another possible scenario could be that the client does not trust a single server with all of its data. In this case, the client might distribute the data across multiple servers. Both of these settings have been studied previously in the literature.

Curtmola *et al.* [27] considered the first of the above two cases. They addressed the problem of storing copies of a single file on multiple servers. This is an attractive solution considering the fact that replication is a fundamental principle in ensuring the availability and durability of data. Their system allows the Verifier to audit a subset of servers even if some of them collude.

On the other hand, Bowers, Juels, and Oprea [19] considered the second of the above two cases. They studied a system where the client's data is distributed and stored on different servers. This ensures that none of the servers has the whole data.

Both of these systems covered one specific instance of the wide spectrum of possibilities when more than one server is involved. For example, none of the works mentioned above address the question of the privacy of data. Both of them argue that, for privacy, the client can encrypt its file before storing it on the servers. These systems are secure only in the computational setting and the privacy guarantee is dependent on the underlying encryption scheme. On the other hand, there are known primitives in the setting of distributed systems, like secret sharing schemes, that are known to be unconditionally secure.

Moreover, we can also utilize cross-server redundancy to get more practical systems. In this chapter, we investigate proof-of-retrievability when there is more than one server.

We define and construct multi-server PoR systems. We motivate and define security of PoR systems in the worst-case and the average-case setting. These two notions capture the scenarios when at least a threshold of servers succeeds with high enough probability (worst-case security) and when the average success probability over all the servers is high enough (average-case security). We also motivate confidentiality of the message as an important requirement in secure cloud storage systems and achieve this without using any encryption scheme.

We start this chapter by giving a formal model of multi-server PoR systems.

6.1 Security Model of Multi-server PoR Systems

The essential components of multi-server PoR (MPoR) systems are natural generalizations of the single-server PoR systems presented in Chapter 3. The first difference is that there are ρ provers and the Verifier might store different messages on each of them. Also, during an audit phase, the Verifier can pick a subset of provers on which it runs the audits. The last crucial difference is that the Extractor has (black-box or non-black-box) access to a subset of proving algorithms corresponding to the provers that the Verifier picked to audit. We detail them below for the sake of completeness.

Let $\mathsf{Prover}_1, \ldots, \mathsf{Prover}_\rho$ be a set of ρ provers and let Verifier be the verifier. The Verifier has a message $m \in \mathcal{M}$ from the message space $\mathcal{M}$ which he redundantly encodes to $M_1, \ldots, M_\rho$.

1. In the keyed setting, the Verifier picks ρ different keys $(K_1, \ldots, K_\rho)$ one for each of the corresponding provers.

2. The Verifier stores M_i on Prover_i. In the case of a keyed scheme, Prover_i may be also given an additional tag S_i generated using the key K_i and M_i.

3. The Verifier stores some sort of information (say a *fingerprint* of the encoded message) that allows him to verify the responses made by the provers.

4. On receiving the encoded message M_i, Prover_i generates a proving algorithm $\mathcal{P}_i$, which it uses to generate its responses during the auditing phase.

5. At any time, the $\mathsf{Verifier}$ picks an index $1 \leq i \leq \rho$ and engages in a challenge-response protocol with Prover_i. In one execution of challenge-response protocol, the $\mathsf{Verifier}$ picks a challenge c and gives it to Prover_i, and the prover responds with ϱ. The $\mathsf{Verifier}$ then verifies the correctness of the response (based on its fingerprint).

6. The success probability $\mathsf{succ}(\mathcal{P}_i)$ is the probability, computed over all the challenges, with which the $\mathsf{Verifier}$ accepts the response sent by Prover_i.

7. The $\mathsf{Extractor}$ is given a subset S of the proving algorithms $\mathcal{P}_1, \ldots, \mathcal{P}_\rho$ (and in the case of a keyed scheme, the corresponding subset of keys $\{K_i : i \in S\}$), and outputs a message $\widehat{m}$. The $\mathsf{Extractor}$ succeeds if $\widehat{m} = m$.

The above framework does not restrict any provers from interacting with other provers when they receive the encoded message. However, we assume that they do not interact *after* they have generated a proving algorithm. If we do not include this restriction, then it is not hard to see that one cannot have any meaningful security. For example, if provers can interact after they receive the encoded message, then it is possible that one prover stores the entire message and the other provers just relay the challenges to this specific prover and relay back its response to the verifier.

In contrast to a single-prover PoR scheme, there are two possible ways in which one can define the security of a multi-prover PoR system. We define them next.

The first security definition corresponds to the "worst case" scenario and is the natural generalization of a single-server PoR system.

Definition 6.1. *A ρ-prover* MPoR *scheme is* (η, ν, τ, ρ)-threshold secure *if there is an* Extractor *which, when given any τ proving algorithms, say $\mathcal{P}_{i_1}, \ldots \mathcal{P}_{i_\tau}$, succeeds with probability at least ν whenever*

$$\mathsf{succ}(\mathcal{P}_j) \geq \eta \qquad \textit{for all } j \in I,$$

where $I = \{i_1, \ldots, i_\tau\}$.

We note that when $\rho = \tau = 1$, we get a standard (η, ν)-PoR system. Moreover, the definition captures the worst-case scenario in the sense that it only guarantees extraction if there exists a set of τ proving algorithms, all of which succeed with high enough probability.

The above definition requires that all the τ servers succeed with high enough probability. On the other hand, it might not be the case that all the proving algorithms of the servers picked by the Verifier succeed with the required probability. In fact, even verifying whether or not all the τ proving algorithms have high enough success probability to allow successful extraction might be difficult (see Section 4.5.2 for more details about this). However, it is possible that some of the proving algorithms succeed with high enough probability to compensate for the failure of the rest of the proving algorithms. For instance, since the provers are allowed to interact before they specify their proving algorithms, it might be the case that the colluding provers decide to store most of the message on a single prover. In this case, even a weaker guarantee that the average success probability is high enough should be enough to guarantee a successful extraction. In other words, it is possible to state (and as we show in this chapter, achieve) a security guarantee with respect to the average case success probability over all the proving algorithms.

Definition 6.2. *A ρ-prover* MPoR *scheme is* (η, ν, ρ)-average secure *if the* Extractor *succeeds with probability at least ν whenever*

$$\frac{1}{\rho}\sum_{i=1}^{\rho} \mathsf{succ}(\mathcal{P}_i) \geq \eta.$$

Note that the average-case secure system reduces to the standard PoR scheme (with $\tau = \rho$) when $\rho = 1$. The following example illustrates that average-case security is possible even when an MPoR system is not possible as per Definition 6.1.

Example 17. Suppose $\eta = 0.7, 0 \leq \nu \leq 1$ and $\rho = 3$. Further, suppose that $\mathsf{succ}(\mathcal{P}_1) = 0.9$, $\mathsf{succ}(\mathcal{P}_2) = 0.6$ and $\mathsf{succ}(\mathcal{P}_3) = 0.6$. Then the hypotheses of Definition 6.1 are not satisfied for $\tau = 2$. So even if the MPoR scheme is (η, ν, τ, ρ)-threshold secure, we cannot conclude that the Extractor will succeed. On the other hand, for the assumed success probabilities, the hypotheses of Definition 6.2 are satisfied. Therefore, if the MPoR scheme is $(0.7, \nu, \tau)$-average secure, the Extractor will succeed.

In this chapter, we are mainly interested in the case when $\nu = 0$ for both the worst-case and the average-case security.

Privacy Guarantee. We mentioned in Chapter 4 that PoR systems were introduced and studied to give assurance of the integrity of the data stored on remote storage. However, the confidentially aspects of data have not been studied formally in the area of cloud-based PoR systems. There have been couple of *ad hoc* solutions that have been proposed in which the messages are encrypted and then stored on the cloud [27]. We believe that, in addition to the standard integrity requirement, privacy of the stored data when multiple provers are involved is also an important requirement. We model the privacy requirement as follows.

Definition 6.3. *An* MPoR *system is called* $\widetilde{\tau}$*-private if no set* $\mathcal{A}$ *of adversarial provers of size at most* $\widetilde{\tau}$ *learns anything about the message stored by the* $\mathsf{Verifier}$*.*

Note that $\widetilde{\tau} = 0$ corresponds to the case when the MPoR system does not provide any confidentiality to the message. The above definition captures the idea that, even if $\widetilde{\tau}$ provers collude, they do not learn anything about the message. We remark that we can achieve confidentiality without encrypting the message by using secret sharing techniques.

6.1.1 Our Contributions

In Section 6.2, we give a construction of an MPoR scheme that achieves worst-case security when the malicious servers are computationally unbounded. Our construction is based on ramp schemes and a single-server PoR scheme. Our construction achieves confidentiality of the message. To exemplify our scheme, we instantiate this scheme with a specific form of ramp scheme.

In Section 6.3, we give a construction of an MPoR scheme that achieves average-case security against computationally unbounded adversaries. For an MPoR system that affords average-case security, we also show that an extension of classical statistical techniques used in Chapter 4 can be used to provide a basis for estimating whether the responses of the servers are accurate enough to allow successful extraction.

One of the main issues in an MPoR system is the maintenance of the stored data, which involves updating. In Section 6.4, we consider this issue. Using our construction in Section 6.2, we extend the construction of single-server dynamic PoR schemes [24] to give an MPoR system that guarantees that an update made by a client is properly registered by the servers. This construction assumes computationally bounded adversaries, due to the limitations of the underlying single-server PoR scheme.

One of the benefits of an MPoR system is that it provides cross-server redundancy. In the past, this feature has been used by Bowers *et al.* [19] to propose a multi-server system called HAIL. We first note that the constructions in Section 6.2 and Section 6.3 do not provide any improvement on the storage overhead of the server or the client. In Section 6.5, we give a construction based on the Shacham-Waters protocol [72] that allows significant reduction of the storage overhead of the client in the multi-server setting.

6.1.2 Comparison with Bowers, Juels, and Oprea

The scheme of Bowers, Juels, and Oprea [19] is closest to this work; however, there are some key differences. We enumerate some of them below:

1. The construction of Bowers, Juels, and Oprea [19] is secure only in the computational setting, while we provide security in the setting of unconditional security.

2. Bowers, Juels, and Oprea [19] use various tools and algorithms to construct their systems, including error-correcting codes, pseudo-random functions, message authentication codes, and universal hash function families. On the other hand, we only use ramp schemes in our constructions, making our schemes far easier to state and analyze.

3. We consider two types of security guarantees — the worst-case scenario and the average-case scenario. Bowers, Juels, and Oprea [19], on the other hand, only consider the worst-case scenario.

4. The construction of Bowers, Juels, and Oprea [19] only aims to protect the integrity of the message, while we consider both the privacy and integrity of the message.

5. We work under a stronger requirement than Bowers, Juels, and Oprea [19] — we require extraction to succeed with probability equal to 1, whereas the extractor of Bowers, Juels, and Oprea [19] succeeds with probability close to 1, depending in part on properties of a certain class of hash functions used in the protocol.

6. Our scheme in the worst case can be easily transformed to a scheme that allows dynamic updates to all the servers. In contrast, the system of Bowers, Juels, and Oprea [19] does not allow efficient updates.

6.2 Worst-case MPoR Based on a Ramp Scheme

In this section, we give our first construction that achieves a worst-case security guarantee. The idea is to use a (τ_1, τ_2, ρ)-ramp scheme in conjunction with a single-server-PoR system. The intuition behind the construction is that the underlying PoR system along with the ramp scheme provides the retrievability guarantee and the ramp scheme provides the confidentiality guarantee.

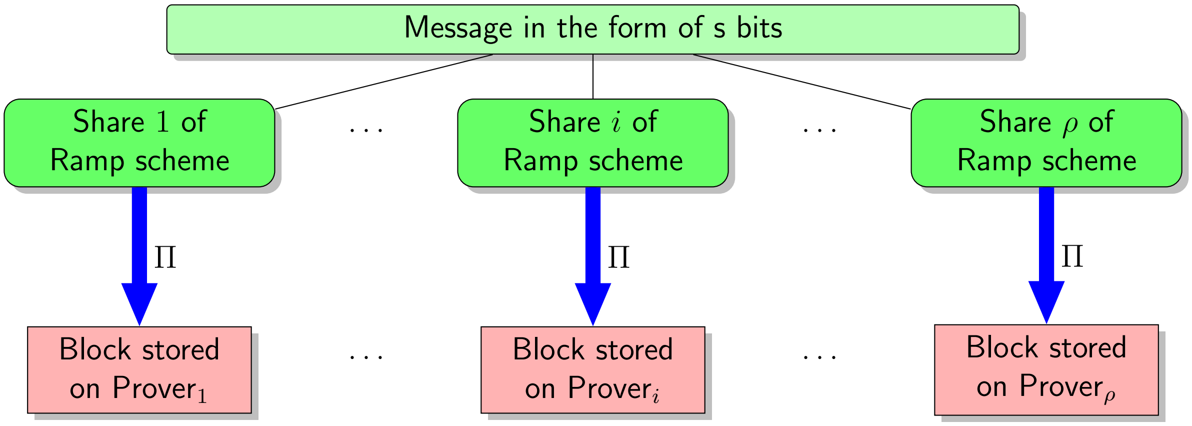
Message in the form of s bits
Share 1 of Ramp scheme
Share i of Ramp scheme
Share ρ of Ramp scheme
. . .
. . .
Π
Π
Π
Block stored on Prover$_1$
Block stored on Prover$_i$
Block stored on Prover$_\rho$

Figure 6.1: Schematic View of a Ramp-MPoR System

We first present a schematic diagram of the working of Ramp-MPoR in Figure 6.1 and illustrate the scheme with the help of following example. We provide the details of the construction in Figure 6.2.

Example 18. Let $\rho = 6$. Suppose the Verifier and the provers use the Linear Combination Scheme as the underlying PoR system Π. The Verifier also picks an encoding function $e(\cdot)$. Let the message to be stored be $(15, 3)$. The Verifier picks $q = 17$ and chooses two random elements $1, 2 \in \mathbb{F}_{17}$ to construct a polynomial $f(x) = 15 + 3x + x^2 + 2x^3$. The Verifier stores $e(4)$ on Prover$_1$, $e(7)$ on Prover$_2$, $e(2)$ on Prover$_3$, $e(1)$ on Prover$_4$, $e(16)$ on Prover$_5$, and $e(8)$ on Prover$_6$.

Let us suppose that the PoR scheme is such that, for a random challenge vector, say $\begin{pmatrix}5, & 2, & 9, & 13, & 5, & 6\end{pmatrix}$, where the i-th entry would be a challenge to Prover$_i$, the corresponding responses of the provers form a vector $\begin{pmatrix}3, & 14, & 1, & 13, & 12, & 14\end{pmatrix}$, where the $\textsc{Resp}_i$ is the correct response of the Prover$_i$. In other words, on challenge 5 to Prover$_1$, the correct response is 3, and so on.

During the audit phase, the Verifier picks any 4 provers. Suppose the Verifier picks Prover$_1$, Prover$_3$, Prover$_4$, and Prover$_6$. The Verifier then sends the corresponding challenges to the provers; i.e., it sends the challenge 5 to Prover$_1$, 9 to Prover$_3$, 13 to Prover$_4$, and 6 to Prover$_6$. If it gets the response 3, 1, 13, and 14 back, it accepts; otherwise, it rejects.

We note one of the possible ways in which the Ramp-MPoR stated in Figure 6.2 can be used in practice. Let m be a message that consists of sk elements from $\mathbb{F}_q$. The Verifier breaks the message into k blocks of length s each. It then invokes a (τ_1, τ_2, ρ)-Ramp scheme

Input: The Verifier gets the message m as input. Let $\mathsf{Prover}_1, \ldots, \mathsf{Prover}_\rho$ be the set of ρ provers.

Initialization Stage. The Verifier performs the following steps for storing the message

1. The Verifier chooses a single-server PoR system Π and a (τ_1, τ_2, ρ)-ramp scheme $\mathsf{Ramp} = (\mathsf{ShareGen}, \mathsf{Reconstruct})$.
2. The Verifier computes ρ shares of the message using the ramp scheme $(m_1, \ldots, m_\rho) \leftarrow \mathsf{ShareGen}(m)$.
3. The Verifier runs ρ independent copies of Π and generates the encoded share $M_i = e(m_i) \in \mathcal{M}$ corresponding to each $1 \leq i \leq \rho$.
4. The Verifier stores M_i on Prover_i.

Challenge Phase: During the audit phase, Verifier picks a prover, Prover_i, and runs the challenge-response protocol of Π with Prover_i.

Figure 6.2: Worst-case Secure MPoR Using a Ramp-scheme (Ramp-MPoR).

on each of these blocks to generate n shares of each of the k blocks. The Verifier then runs a PoR scheme Π to compute the encoded message to be stored on each of the servers by encoding its k shares, one corresponding to each of the k blocks.

We prove the following security result for the MPoR scheme presented in Figure 6.2.

Theorem 6.1. *Let Π be an $(\eta, 0, 1, 1)$-threshold secure* MPoR *with a response code of Hamming distance $\widetilde{\mathsf{d}}$ and the size of challenge space γ and let* $\mathsf{Ramp} = (\mathsf{ShareGen}, \mathsf{Reconstruct})$ *be a (τ_1, τ_2, ρ)-ramp scheme. Then* Ramp-MPoR*, defined in Figure 6.2, is an* MPoR *system with the following properties:*

1. *Privacy:* Ramp-MPoR *is τ_1-private.*
2. *Security:* Ramp-MPoR *is $(\eta, 0, \tau_2, \rho)$-threshold secure, where $\eta = 1 - \widetilde{d}/2\gamma$.*

Proof. The privacy guarantee of Ramp-MPoR is straightforward from the privacy property of the underlying ramp scheme.

For the security guarantee, we need to demonstrate an $\mathsf{Extractor}$ that outputs a message $\widehat{m} = m$ if there exists a set of τ_2 servers that succeed with probability at least $\eta = 1 - \widetilde{\mathsf{d}}/2\gamma$. The description of our $\mathsf{Extractor}$ is as follows:

1. The $\mathsf{Extractor}$ chooses τ_2 provers and runs the extraction algorithm of the underlying single-server PoR system on each of these provers. In the end, it outputs $\widehat{M}_{i_j}$ for the corresponding provers Prover_{i_j}. It defines $\mathcal{S} \leftarrow \{\widehat{M}_{i_1}, \ldots, \widehat{M}_{i_{\tau_2}}\}$.

2. The $\mathsf{Extractor}$ invokes the $\mathsf{Reconstruct}$ algorithm of the underlying ramp scheme with the elements of $\mathcal{S}$. It outputs whatever $\mathsf{Reconstruct}$ outputs.

Now note that the $\mathsf{Verifier}$ interacts with every Prover_i independently. We know from the security of the underlying single-server-PoR scheme (Theorem 4.1) that there is an extractor that always outputs the encoded message whenever $\mathsf{succ}(\mathcal{P}_i) \geq \eta$. Therefore, if all the τ_2 chosen proving algorithms succeed with probability at least η, then the set $\mathcal{S}$ will have τ_2 correct shares. From the correctness of the $\mathsf{Reconstruct}$ algorithm, we know that the message output in the end by the $\mathsf{Extractor}$ will be the message m. □

As a special case of the above, we get a simple MPoR system which uses a *replication code*. This is the setting considered by Curtmola *et al.* [27]. A replication code has an encoding function $\mathsf{Enc} : \Lambda \rightarrow \Lambda^\rho$ such that $\mathsf{Enc}(x) = (\underbrace{x, x, \ldots, x}_{\rho \text{ times}})$ for any $x \in \Lambda$. It should be noted that a ρ-replication code is a $(0, 1, \rho)$-ramp scheme. We call a $\mathsf{Ramp\text{-}MPoR}$ scheme based on a replication code a $\mathsf{Rep\text{-}MPoR}$. The schematic description of the scheme is presented in Figure 6.3 and the scheme is presented in Figure 6.4. A simple corollary to Theorem 6.1 is the following.

Corollary 6.2. *Let* Π *be a* $(\eta, 0, 1, 1)$-MPoR *system with a response code of Hamming distance* $\widetilde{\mathsf{d}}$ *and the size of challenge space* γ*. Then* $\mathsf{Ramp\text{-}MPoR}$*, formed by instantiating* $\mathsf{Ramp\text{-}MPoR}$ *with the replication code ramp scheme, is an* MPoR *system with the following properties:*

1. *Privacy: It is* 0*-private.*

2. *Security: It is* $(\eta, 0, 1, \rho)$*-threshold secure.*

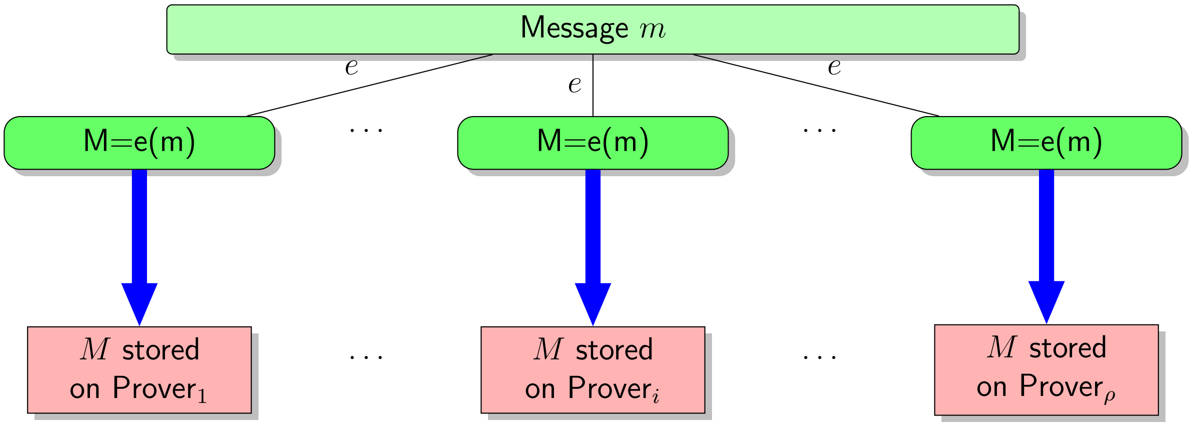

Figure 6.3: Schematic View of Rep-MPoR

6.3 Average-case Secure MPoR System

In general, it is not possible to verify with certainty whether the success probability of a proving algorithm is above a certain threshold; therefore, in that case, it is unclear how the Extractor would know which proving algorithms to use for the extraction purpose in Section 6.2. In this section, we analyze the average-case security properties of the replication code based scheme, Rep-MPoR, described in the last section. This allows us an alternative guarantee that allows successful extraction where the extractor need not worry whether a certain proving algorithm succeeds with high enough probability or not.

Recall the scenario introduced in Example 17. Here we assumed $\mathsf{succ}(\mathcal{P}_1) = 0.9$, $\mathsf{succ}(\mathcal{P}_2) = 0.6$, and $\mathsf{succ}(\mathcal{P}_3) = 0.6$ for three provers. Suppose that successful extraction for a particular prover $\mathcal{P}_i$ requires $\mathsf{succ}(\mathcal{P}_2) \geq 0.7$. Then extraction would work on only one of these three provers. On the other hand, suppose we have an average-case secure MPoR in which extraction is successful if the average success probability of the three provers is at least 0.7. Then the success probabilities assumed above would be sufficient to guarantee successful extraction.

Theorem 6.3. *Let* Π *be a single-server* PoR *system with a response code of Hamming distance* $\widetilde{\mathsf{d}}$ *and the size of challenge space* γ. *Then* Rep-MPoR, *defined in Figure 6.4, is an* MPoR *system with the following properties:*

1. *Privacy:* Rep-MPoR *is* 0*-private.*
2. *Security:* Rep-MPoR *is* $(1 - \widetilde{\mathsf{d}}/2\gamma, 0, \rho)$*-average secure.*

Input: The verifier Verifier gets the message m as input. Let $\mathsf{Prover}_1, \dots, \mathsf{Prover}_\rho$ be the set of ρ-provers.

Initialization Stage. The Verifier performs the following steps for storing the message

1. The Verifier chooses a single-server PoR system Π.
2. Using the encoding scheme of Π, the Verifier generates the encoded message $M = e(m) \in \mathcal{M}$ for $1 \leq i \leq n$.
3. The Verifier stores the message M on all Prover_i for $1 \leq i \leq n$.

Challenge Phase: During the audit phase, Verifier runs the challenge-response protocol of Π independently on each server.

Figure 6.4: Average-case Secure MPoR (Rep-MPoR).

Proof. Since the message is stored in its entirety on each of the servers, there is no confidentiality.

For the security guarantee, we need to demonstrate an Extractor that outputs a message $\widehat{m} = m$ if the average success probability of all the provers is at least $\eta = 1 - \widetilde{\mathsf{d}}/2\gamma$. The description of our Extractor is as follows:

1. For all $1 \leq i \leq n$, use $\mathcal{P}_i$ to compute the vector $R_i = (r_c^{(i)} : c \in \Gamma)$, where $r_c^{(i)} = \mathcal{P}_i(c)$ for all $c \in \Gamma$ (i.e., for every c, $r_c^{(i)}$ is the response computed by $\mathcal{P}_i$ when it is given the challenge c),
2. Compute R as a concatenation of $R_1, \dots, R_\rho$ and find $\widehat{M} := \left(\widehat{M}_1, \dots, \widehat{M}_\rho\right)$ so that $\mathsf{dist}(R, r^{\widehat{M}})$ is minimized, and
3. Compute $m = e^{-1}(\widehat{M})$.

Now note that Verifier interacts with each Prover_i independently and Extractor uses the challenge-response step with independent challenges. Let $\eta_1, \dots, \eta_\rho$ be the success probabilities of the ρ proving algorithms. Let $\bar{\eta}$ be the average success probability over all the servers and challenges. Therefore, $\bar{\eta} = \rho^{-1} \sum_{i=1}^{\rho} \eta_i$.

First note that, in the case of Figure 6.4, the response code is of the form

$$\left\{ \underbrace{(r, r, \ldots, r)}_{\rho \text{ times}} : r \in \mathcal{R} \right\}.$$

It is easy to see that the distance of the response code is $\rho\widetilde{\mathsf{d}}$ and the length of a challenge is $\rho\gamma$. From the definition of the extractor and Theorem 4.1, it follows that the extraction succeeds if

$$\frac{\eta_1 + \ldots + \eta_\rho}{\rho} = \bar{\eta} \geq 1 - \frac{\widetilde{\mathsf{d}}}{2\gamma}.$$

□

6.3.1 Hypothesis Testing for Rep-MPoR

For the purposes of auditing whether a file is being stored appropriately, it is necessary to have a mechanism for determining whether the success probability of a prover is sufficiently high. In the case of replication code based MPoR with worst-case security, we are interested in the success probabilities of individual provers, and the analysis can be carried out as detailed in Chapter 4. In the case of Rep-MPoR, however, we wish to determine whether the *average* success probability of the set of provers $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_\rho\}$ is at least η. This amounts to distinguishing the null hypothesis

$$H_0 : \mathsf{avg\text{-}succ}(\mathcal{P}_i) < \eta;$$

from the alternative hypothesis

$$H_1 : \mathsf{avg\text{-}succ}(\mathcal{P}_i) \geq \eta.$$

Suppose we send c challenges to each server. If a given server $\mathcal{P}_i$ has success probability $\mathsf{succ}(\mathcal{P}_i)$, then the number of correct responses received follows the binomial distribution $\mathsf{B}(c, \mathsf{succ}(\mathcal{P}_i))$. If the success probabilities $\mathsf{succ}(\mathcal{P}_i)$ were the same for each server, then the sum of the number of successes over all the servers would also follow a binomial distribution. However, we are also interested in the case in which these success probabilities differ, in which case the total number of successes follows a *poisson-binomial distribution,* which is more complicated to work with. In order to establish a test that is conceptually and computationally easier to apply, we will instead rely on the observation that, in cases where the average success probability is high enough to permit extraction, the failure rates of the servers are relatively low.

For a given server $\mathcal{P}_i$, let $f_i = 1 - \mathsf{succ}(\mathcal{P}_i)$ denote the probability of failure. For r challenges, the number of failures follows the binomial distribution $\mathsf{B}(c, f_i)$. Provided that r is sufficiently large and f_i is sufficiently low, then $\mathsf{B}(c, f_i)$ can be approximated by the *poisson distribution* $\mathsf{Pois}(cf_i)$. The poisson distribution $\mathsf{Pois}(\lambda)$ is used to model the scenario where discrete events are occurring independently within a given time period with an expected rate of λ events during that period. The probability of observing k events within that period is given by

$$P(k) = \frac{e^{-\lambda}\lambda^k}{k!}.$$

The mean and the variance of $\mathsf{Pois}(\lambda)$ is equal to λ. For our purposes, the advantage of using this approximation is the fact that the sum of ρ independent variables following the poisson distributions $\mathsf{Pois}(\lambda_1), \mathsf{Pois}(\lambda_2), \ldots, \mathsf{Pois}(\lambda_\rho)$ is itself distributed according to the poisson distribution $\mathsf{Pois}(\lambda_1 + \lambda_2 + \cdots + \lambda_\rho)$, even when the λ_i all differ. This can be seen because of the following reason. The moment generating function of a Poisson distribution is $e^{-\lambda(1-k)}$. Therefore, the sum of two independent random variables following the Poisson distributions $\mathsf{Pois}(\lambda_1)$ and $\mathsf{Pois}(\lambda_2)$ has the moment generating function $e^{-\lambda_1(1-k)}e^{-\lambda_2(1-k)}$. This is same as the moment generating function of $\mathsf{Pois}(\lambda_1 + \lambda_2)$.

In the case where the average failure probability is low, the distribution $\mathsf{Pois}(c(f_1 + f_2 + \cdots + f_\rho))$ should provide a reasonable approximation to the actual distribution of the total number of failed challenges.

Example 19. To demonstrate the appropriateness of the Poisson approximation for this application, suppose we have five servers, whose failure probabilities are expressed as $\mathbf{f} = (f_1, f_2, \ldots, f_5)$. Let t be the number of trials per server and b the total number of observed failures out of the $5t$ trials. Table 6.1 gives both the exact cumulative probability $\mathsf{Pr}[B \le b]$ of observing up to b failures, and the Poisson approximation $\mathsf{Pr}_{\mathsf{Pois}}[B \le b]$ of this cumulative probability, for a range of values for $\mathbf{f}$.

Table 6.1: Comparison Between Exact Cumulative Probability and Approximation by Poisson Distribution

$\mathbf{f} = (0.1, 0.1, 0.1, 0.1, 0.1)$

t	b	$\mathsf{Pr}[B \le b]$	$\mathsf{Pr}_{\mathsf{Pois}}[B \le b]$
200	5	$2.556545692 \times 10^{-38}$	$3.261456422 \times 10^{-36}$
200	10	$1.450898832 \times 10^{-32}$	$1.137687971 \times 10^{-30}$

Continued on next page ...

Table 6.1 — continued from previous page

200	50	$5.995167631 \times 10^{-9}$	$2.401592276 \times 10^{-8}$
200	100	0.5265990813	0.5265622074
100	0	$1.322070819 \times 10^{-23}$	$1.928749864 \times 10^{-22}$
100	5	$6.272915577 \times 10^{-17}$	$5.567756307 \times 10^{-16}$
100	10	$1.135691814 \times 10^{-12}$	$6.450152972 \times 10^{-12}$
100	15	$1.662665039 \times 10^{-9}$	$6.357982164 \times 10^{-9}$
100	20	$4.557480806 \times 10^{-7}$	0.000001235187232
200	0	$1.747871252 \times 10^{-46}$	$3.720076039 \times 10^{-44}$
200	5	$2.556545692 \times 10^{-38}$	$3.261456422 \times 10^{-36}$
200	10	$1.450898832 \times 10^{-32}$	$1.137687971 \times 10^{-30}$
200	15	$6.757345217 \times 10^{-28}$	$3.340076418 \times 10^{-26}$
200	20	$5.962487876 \times 10^{-24}$	$1.905558774 \times 10^{-22}$
500	20	$1.240463044 \times 10^{-84}$	$1.084188102 \times 10^{-79}$
500	25	$3.140367419 \times 10^{-79}$	$1.697380630 \times 10^{-74}$
500	30	$2.935666094 \times 10^{-74}$	$9.912214279 \times 10^{-70}$
500	35	$1.193158517 \times 10^{-69}$	$2.542280876 \times 10^{-65}$
500	40	$2.369596756 \times 10^{-65}$	$3.218593843 \times 10^{-61}$

$\mathbf{f} = (0.01, 0.01, 0.01, 0.01, 0.01)$

t	b	$\mathsf{Pr}[B \leq b]$	$\mathsf{Pr}_{\mathsf{Pois}}[B \leq b]$
200	5	0.06613951161	0.06708596299
200	10	0.5830408032	0.5830397512
200	20	0.9985035184	0.9984117410
200	50	≈ 1	≈ 1

$\mathbf{f} = (0.2, 0.01, 0.02, 0.03, 0.04)$

t	b	$\mathsf{Pr}[B \leq b]$	$\mathsf{Pr}_{\mathsf{Pois}}[B \leq b]$
200	5	$9.651421837 \times 10^{-22}$	$6.180223643 \times 10^{-20}$
200	10	$5.539867010 \times 10^{-17}$	$1.744235672 \times 10^{-15}$
200	20	0.09020056729	0.1076778797
200	50	0.9999999198	0.9999991415

$\mathbf{f} = (0.01, 0.01, 0.03, 0.04, 0.05)$

t	b	$\mathsf{Pr}[B \leq b]$	$\mathsf{Pr}_{\mathsf{Pois}}[B \leq b]$
200	5	$8.312224722 \times 10^{-8}$	$1.196952269 \times 10^{-7}$
200	10	0.00006809921297	0.00008550688580
200	20	0.06901537242	0.07274102693
200	50	0.9999582547	0.9999397284

Continued on next page . . .

Table 6.1 — continued from previous page

$\mathbf{f} = (0.1, 0.1, 0.1, 0.1, 0.1)$

t	b	$\mathsf{Pr}[B \leq b]$	$\mathsf{Pr}_{\mathsf{Pois}}[B \leq b]$
20	0	0.00002656139888	0.00004539992984
20	5	0.05757688648	0.06708596299
20	10	0.5831555123	0.5830397512
20	15	0.9601094730	0.9512595983
20	20	0.9991924263	0.9984117410
40	0	$7.055079108 \times 10^{-10}$	$2.061153629 \times 10^{-9}$
40	5	0.00003871193246	0.00007190884076
40	10	0.008071249954	0.01081171886
40	15	0.1430754340	0.1565131351
40	20	0.5591747822	0.5590925860
100	20	$4.557480806 \times 10^{-7}$	0.000001235187232
100	25	0.00003540113222	0.00007160717427
100	30	0.001002549708	0.001594027332
100	35	0.01231948910	0.01621388016
100	40	0.07508928967	0.08607000083
20	0	0.3660323413	0.3678794412
20	5	0.9994654657	0.9994058153
20	10	0.9999999939	0.9999999900
20	15	1.000000000	1.000000000
20	20	1.000000000	1.000000000
40	0	0.1339796748	0.1353352833
40	5	0.9839770930	0.9834363920
40	10	0.9999931182	0.9999916922
40	15	0.9999999996	1.000000000
40	20	0.9999999999	1.000000000
100	20	0.9999999367	0.9999999198
100	25	0.9999999999	1.000000001
100	30	0.9999999999	1.000000001
100	35	0.9999999999	1.000000001
100	40	0.9999999999	1.000000001

$\mathbf{f} = (0.02, 0.0075, 0.0075, 0.0075, 0.0075)$

t	b	$\mathsf{Pr}[B \leq b]$	$\mathsf{Pr}_{\mathsf{Pois}}[B \leq b]$
20	0	0.08936904038	0.09536916225

Continued on next page . . .

Table 6.1 — continued from previous page

20	5	0.9712600336	0.9672561739
20	10	0.9999843669	0.9999642885
20	15	0.9999999995	0.9999999958
20	20	1.000000000	1.000000000
40	0	0.007986825382	0.009095277109
40	5	0.6699740391	0.6684384858
40	10	0.9927425867	0.9909776597
40	15	0.9999835852	0.9999661876
40	20	0.9999999935	0.9999999715
100	20	0.9999999935	0.9999999715
100	25	0.9999999998	1.000000001
100	30	0.9999999998	1.000000001
100	35	0.9999999998	1.000000001
100	40	0.9999999998	1.000000001

As an example of using the given formula to calculate a confidence interval, suppose we do 200 trials on each of five servers (so there are 1000 trials in total) and we observe 50 failures in total. Then the resulting confidence interval is $[0, 63.29)$. Suppose we wish to know whether the success probability is at least $\eta = 0.9$. We have $(1 - 0.9) \times 1000 = 100$. This is outside of that interval, and hence we conclude there is enough evidence to reject H_0 at the 95% significance level. However, to test whether the success probability was greater than 0.95 we see that $(1 - 0.95) \times 1000 = 50$. Since 50 lies within the interval, we conclude there is insufficient evidence to reject H_0 at the 95% significance level.

Let b denote the number of incorrect responses we have received from the $c\rho$ challenges given to the provers. Suppose that H_0 is true, so that the expected number of failures is at least $\eta\rho c$. Based on our approximation, the probability that the number of failures is at most b is at most

$$\sum_{i=0}^{b} \frac{e^{-\eta\rho c}(\eta\rho c)^i}{i!}.$$

If this probability is less than 0.05, we reject H_0 and accept the alternative hypothesis. However, if the probability is greater than 0.05, then there is not enough evidence to reject H_0 at the 5% significance level, and so we continue to suspect that the file is not being stored appropriately.

We can express this test neatly using a *confidence interval*. We define a 95% upper confidence bound by

$$\lambda_U = \inf\left\{\lambda \middle| \sum_{i=0}^{b} \frac{e^{-\lambda}\lambda^i}{i!} < 0.05\right\}.$$

This represents the smallest parameter choice for the Poisson distribution for which the probability of obtaining b or fewer incorrect responses is less than 0.05. Then $[0, \lambda_U)$ is a 95% confidence interval for the mean number of failures, so we reject H_0 whenever ηnr lies outside this interval. The value of λ_U can be determined easily by exploiting a connection with the chi-squared distribution [82]: we have that

$$\sum_{i=0}^{b} \frac{e^{-\lambda}\lambda^i}{i!} = \Pr(\chi^2_{2b+2} > 2\lambda),$$

and so the appropriate value of λ_U can readily be obtained from a table for the chi-squared distribution.

6.4 Maintaining an MPoR scheme: Dynamic Updates

One of the desirable features of cloud-based storage is to allow efficient updates to messages stored on the cloud at any given time. In this section, whenever we update the message, we mean that the original message m is changed to some m' without changing its size. Note that we do not assume that the Verifier has a copy of its message that it stored on the servers.

We start by discussing the problems in making efficient updates to the encoded message in a PoR scheme. This will also help us in arguing the case in the multi-server setting. At a very high level, all PoR constructions share essentially the same common structure. The Verifier stores some redundant encoding of its message under a code on the Prover, ensuring that the Prover must delete a significant fraction of the encoding before losing any fraction of the actual message. During an audit, the Verifier then checks a few random locations of the encoding, so that a Prover who deleted a significant fraction will get caught with overwhelming probability. This construction makes it very difficult to accommodate efficient updates. For example, if we need to change a single location in m, then we may have to change a constant fraction of the encoded message. In the rest of this subsection, we use the letter m to denote the original message and M to denote the encoded message.

In practice, the message m is usually broken down into n blocks, then each block is encoded using an encoding function $e(\cdot)$, and the encoded message is stored on the prover.[1] This is to improve the communication cost during an audit phase. We denote the encoded M as an n-block message $(M[1], \dots, M[n])$. In this setting, an update means a single block of the message is changed. In the multi-server setting, a message is shared among ρ provers. When an encoded message M is shared between the provers as $M_1, \dots, M_\rho$, we write the blocks of the encoded message stored on Prover_i as $M_i[1], \dots, M_i[n]$.

6.4.1 Constructing a Dynamic-MPoR

In this section, we give a construction of an MPoR scheme that allows dynamic updates. We use a dynamic-PoR scheme that is secure in the single-server setting and use it to construct a dynamic-MPoR. There are a few things to take into account when we try to extend a dynamic-PoR scheme in a single server setting to a multi-server system. It is clear that we have to divide the encoded message into blocks and share it among the provers, but we need to make sure that any update to a single message block does not require updates on too many provers and that the number of required updates on every prover is also small. The first restriction is to keep the communication complexity small during an audit and the second one is to justify the use of a dynamic-PoR system. For example, if a single update to a message results in an update to many encoded message blocks, a Verifier might as well wait for a number of updates. The Verifier could then download the whole encoded message, perform all the updates at the same time, and restore the changed encoded message.

The main idea that we use to extend the single-server dynamic-PoR systems to the multi-server setting is as follows. Suppose the message and the encoded message are n blocks long, where each message block is an element of $(\mathbb{F}_q)^{\mathsf{s}}$. We use the Ramp-MPoR scheme from Figure 6.2. In the notation of the Figure 6.2, we instantiate Π and the Ramp scheme appropriately. We use a dynamic PoR system for Π and a linear secret sharing scheme for the Ramp scheme. Note that any linear code gives a Ramp scheme (Theorem 2.6) and also an LSSS (Construction 2). We can therefore pick an appropriate $[N, k, d]_q$-linear code. As we shall see later, we need some more properties, in addition to the linearity property of a code, to make our scheme efficient. The choice of Π is more straightforward. Note that, since all the known constructions of dynamic-PoR schemes

[1] The encoding function $e(\cdot)$ is in practice some form of redundant encoding which maps from a smaller bit-length space to a larger bit-length space. For example, one possible encoding function could be $e : \mathbb{F}_p \to \mathbb{F}_q$, where $q > p$.

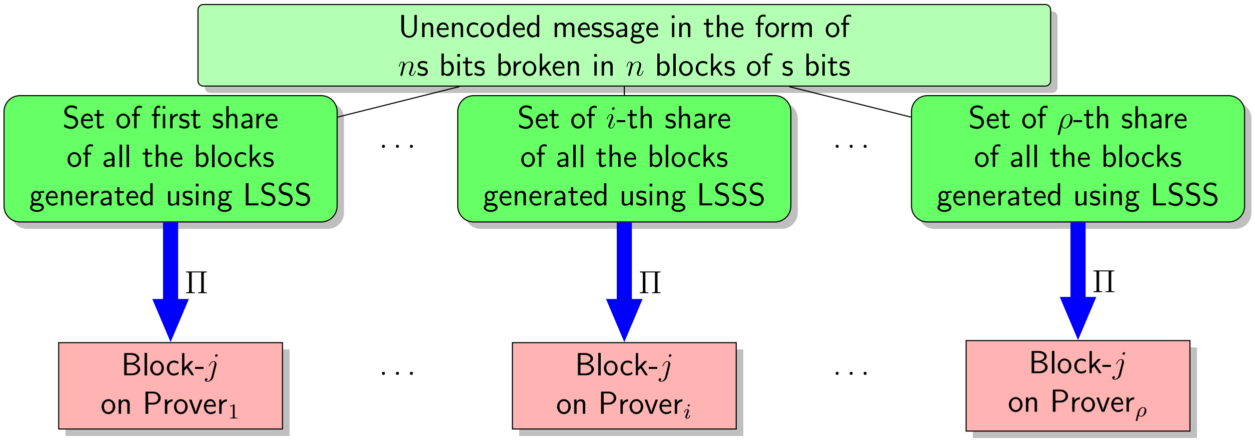
Unencoded message in the form of
ns bits broken in n blocks of s bits
Set of first share of all the blocks generated using LSSS
. . .
Set of i-th share of all the blocks generated using LSSS
. . .
Set of ρ-th share of all the blocks generated using LSSS
Π
Π
Π
Block-j on Prover$_1$
. . .
Block-j on Prover$_i$
. . .
Block-j on Prover$_\rho$

Figure 6.5: Schematic View of Dynamic-MPoR System where Π is a single-server dynamic-PoR system

are secure only against computationally bounded adversaries, our final construction is also secure (i.e., satisfies Definition 3.1) only against computational adversaries. However, we do not make any additional computational assumptions and the privacy guarantee is still information-theoretic. For example, the construction of Chandran *et al.* [24] assumes a secure *message authentication code* (MAC); therefore, our construction for Dynamic-MPoR is also only secure under the assumption that there is a secure MAC.

At a high level, the Dynamic-MPoR given in Figure 6.6 works as follows (see the schematic diagram in Figure 6.5). The Verifier receives an n-block message m. We assume that each block consists of s elements each of which are in the finite field $\mathbb{F}_q$. When this is the case, we say that each block is an element of $(\mathbb{F}_q)^{\mathsf{s}}$. It first picks a linear code that is also a (τ_1, τ_2, ρ)-LSSS. It computes the shares for the provers for each of the n blocks of m using the share generation algorithm of a (τ_1, τ_2, ρ)-LSSS scheme. It then stores these blocks on the ρ servers. Therefore, every prover has n blocks, each of which is an element of $\mathbb{F}_q$. During an update, if the i-th block of m has to be updated from m_i to $\widetilde{m}_i = m_i + \xi$ for some $1 \leq i \leq k$, the Verifier computes the share corresponding to a message which has zero entries everywhere except for ξ at the i-th position. Let the shares generated be $\Xi \in (\mathbb{F}_q)^\rho$. The Verifier now updates only those provers Prover$_j$ for which the j-th entry of the codeword, Ξ_j is non-zero, for $1 \leq j \leq n$, using the update algorithm of the underlying single-server dynamic-PoR scheme.

We prove the following result for the scheme presented in Figure 6.6. It should be noted that all existing dynamic-PoR schemes are secure only in the computational setting. This

Input: The Verifier has the message $m = (m[1], \ldots, m[n])$ where each $m[i] \in (\mathbb{F}_q)^{\mathsf{s}}$. Let $\mathsf{Prover}_1, \ldots, \mathsf{Prover}_\rho$ be the set of ρ provers.

Initialization Stage. Let Π denote a single-prover dynamic PoR scheme and let (τ_1, τ_2, ρ)-LSSS denote a linear secret sharing scheme based on a $[\rho + \mathsf{s}, k]_q$ linear code with $\mathsf{s} \leq \mathsf{d}^\perp - 2$. Let $\mathsf{Enc}(\cdot)$ be the encoding function as defined in Section 2.4.1. The Verifier now performs the following steps to encode and store the message m.

1. For $1 \leq i \leq n$, use Enc to compute ρ shares of $m[i]$ as follows: $(M_1[i], \ldots, M_\rho[i]) = \mathsf{Enc}(m[i])$, where $m[i] = (m_1[i], \ldots, m_{\mathsf{s}}[i]) \in (\mathbb{F}_q)^{\mathsf{s}}$.
2. For $1 \leq j \leq \rho$, use Π to encode $(M_j[1], \ldots, M_j[n])$ and store the result on Prover_j.

Challenge Phase: The Verifier uses the challenge-response algorithm of Π to audit the provers of its choice.

Update Phase: Suppose the Verifier wishes to update the j-th position of the i-th block of the message to $\widetilde{m}[i] = m[i] + \xi$ for some $1 \leq j \leq \mathsf{s}, 1 \leq i \leq n$, where $\xi \in (\mathbb{F}_q)^{\mathsf{s}}$ has hamming weight 1 and $\xi_j \neq 0$. The Verifier does the following:

1. Compute the shares corresponding to ξ as follows: $\Xi = \mathsf{Enc}(\xi)$.
2. The Verifier uses the update procedure of Π to update the message encoding by adding Ξ_h to $M_h[i]$ for all $h \in \{d : \Xi_d \neq 0\}$.

Figure 6.6: Dynamic MPoR Using Linear Secret Sharing Scheme (Dynamic-MPoR)

is the reason why we have a security parameter in our scheme.

Theorem 6.4. *Let ζ be the maximum number of non-zero entries in the rows of the generator matrix used to generate the code used to define the (τ_1, τ_2, ρ)-LSSS. Let κ be the security parameter of a single-server dynamic PoR system, Π, with the following properties: the storage requirement on the Prover is $\mathsf{S}_{\mathsf{Prover}}$, on the verifier is $\mathsf{S}_{\mathsf{Verifier}}$, a write update takes w time, accessing any entry on the prover takes r time, and an audit protocol takes a time. Then, Dynamic-MPoR, as presented in Figure 6.6, provides the following guarantees:*

1. *Storage Requirements: The total storage required on every Prover_i is $\mathsf{S}_{\mathsf{Prover}}$ for $1 \leq i \leq \rho$, and Verifier needs $\rho \mathsf{S}_{\mathsf{Verifier}}$ storage.*

2. *Efficiency: The write update takes $O(\zeta\mathsf{w})$ time, accessing an entry at any* Prover*'s storage takes $O(\zeta\mathsf{r})$ time, and an audit protocol takes $O(\zeta\mathsf{a})$.*

3. *Security: If Π is $(\eta, \nu/\tau_2, 0, 1)$-secure, then* Dynamic-MPoR *is $(\eta, \nu, \tau_2, \rho)$-threshold secure.*

4. *Privacy:* Dynamic-MPoR *is τ_1-private.*

Before proving the theorem, we make an important remark. In Enc, as defined in Section 2.4.1, the encoding function picks $k - \mathsf{s}$ random chosen values and then forms a k-block message by concatenating ξ with these uniformly and randomly chosen values. However, in the proof of the theorem, we fix these random values to be all-zeroes vector. This would allow the provers, which are updated, to learn which position of the original file has changed, but they do not learn the final value of the message. Therefore, the privacy of the system is still maintained, in accordance with Definition 6.3.

Proof. The storage requirement is fairly straightforward from the description given in Figure 6.6. For the efficiency guarantee, we first determine the time required by a single update. Note that in a Dynamic-MPoR, Prover_i has to be updated if $\Xi_i \neq 0$. One natural and practical requirement would be to minimize the number of provers that need to be updated. In the next part of this section, we analyze the number of changes in the codewords required.

Efficiency Guarantee. Let m be the original message of the Verifier and let $\widetilde{m}$ be the modified message. Without loss of generality, we assume that $\mathsf{dist}(m, \widetilde{m}) = 1$, and m and $\widetilde{m}$ differ in the i-th position. Let $m_i - \widetilde{m}_i = \xi \in \mathbb{F}_q$. First note that an alternate way to see any linear secret sharing scheme is in the form of an error correcting code. Let $\mathbf{G}$ be the generator matrix in the standard form corresponding to the error correcting code on which the LSSS scheme is based. We pick $\mathbf{x} = \xi \parallel \mathbf{0}^{k-\mathsf{s}}$ and compute $\mathbf{c} = \mathbf{x}\mathbf{G}$. We then set Ξ to the last ρ entries of $\mathbf{c} = \mathbf{x}\mathbf{G}$. That is, the hamming weight of Ξ is the same as the number of non-zero entries in the i-th row of the matrix $\mathbf{G}$. For a general linear code, there is no bound on this value. On the other hand, if we use a linear code whose generator matrix has small row sparsity [43], we can improve the update efficiency significantly. For this, we can use various constructions, depending on the structure of the finite field $\mathbb{F}_q$, mentioned in Section 2.2.2. For example, we can use a code whose parity check matrix is the one defined by Bennatan and Burshtein [13]. On the other hand, if we have $\mathbb{F}_{q^s}$ for an integer s, then we can use a code whose parity check matrix is the one defined by Kou, Lin, and Fossorier [55] (Theorem 2.3). Let ζ be the row sparsity of the generator matrix used in the construction of LSSS. From the above discussion, it is clear that we need to update at most ζ provers. From the hypothesis of the theorem, a single update on any prover takes $O(\mathsf{w})$ time; therefore, it takes $O(\zeta\mathsf{w})$ total time to update all the servers. This restricts our access time while auditing a particular entry of the encoded message and increases the total storage on provers to $O(\zeta\mathsf{S}_{\mathsf{Prover}})$.

Privacy and Security Proof. For the privacy guarantee, recall from Section 2.4 that any $[N, k]$-linear code with distance $\mathbf{d}$ and dual distance $\mathbf{d}^\perp$ is an (τ_1, τ_2, ρ)-ramp scheme scheme, where $N = \rho + \mathsf{s}$, $\tau_1 = N - \mathbf{d} + 1$ and $\tau_2 = \mathbf{d}^\perp + 2$. Therefore, part 4 follows from the secrecy guarantee of the underlying LSSS scheme.[2]

The security guarantee is analogous to that of Theorem 6.1, except that we need to be careful because $\nu \neq 0$ for the dynamic-PoR schemes we are going to use. Let $\mathcal{P}_1, \ldots, \mathcal{P}_\rho$ be the proving algorithms for $\mathsf{Prover}_1, \ldots, \mathsf{Prover}_\rho$, respectively. For the security guarantee, we have to show an Extractor that outputs m if $\mathsf{succ}(\mathcal{P}_i) > \eta$ for at least τ_2 proving algorithms. The description of the Extractor is just as in the case of Theorem 6.1:

1. Extractor, on input a set of τ_2 proving algorithms $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_{\tau_2}}$, runs the extractor of the underlying single-server PoR system Π with each of $\mathcal{P}_{i_j}$ for $1 \leq j \leq \tau_2$ to output $\widehat{M}_{i_1}, \ldots, \widehat{M}_{i_{\tau_2}}$. Define $\mathcal{S} \leftarrow \{\widehat{M}_{i_1}, \ldots, \widehat{M}_{i_{\tau_2}}\}$.
2. Extractor invokes Reconstruct algorithm of the underlying LSSS with the the elements of the set $\mathcal{S}$. It outputs whatever Reconstruct outputs.

First note that the Verifier interacts with Prover_i independently for all $i \in \{i_1, \ldots, i_{\tau_2}\}$. Since the Verifier uses independent instances of $(\eta, \nu/\tau_2, 0, 1)$-secure PoR systems Π, we know from the security of Π that there is an extractor that outputs the encoded message with probability at least $1 - \nu/\tau_2$ whenever $\mathsf{succ}(\mathcal{P}_i) \geq \eta$. Therefore, if τ_2 proving algorithms succeeds with probability at least η, then using the union bound, the set $\mathcal{S}$ will have at least τ_2 correct shares with probability $1 - \nu$. From the correctness of the Reconstruct algorithm, we know that the message output in Step 3 of Extractor will be the message m. This completes the proof of the theorem. □

The scheme presented in Figure 6.6 is in its most general form. In order to instantiate the scheme and get concrete bounds on the security, privacy, and efficiency guarantees, we need to do the following:

1. Pick a dynamic-PoR system which is secure in the single server setting.

[2] The distance is easier to calculate from the column sparsity of the parity check matrix; however, the dual distance needs more work. We perform the computations for two schemes later in this subsection.

2. Pick a linear code which provides the required privacy and efficiency guarantee.

One could pick different constructions of linear codes which satisfies the required properties to instantiate the LSSS used in Figure 6.6. Also, we can pick any of the three existing dynamic-PoR constructions [23, 24, 75] to instantiate Π. Each of these choices would give different security, privacy, and efficiency guarantees for the scheme presented in Figure 6.6. In the next two subsections, we discuss these possible choices in more detail.

6.4.2 Choosing a Dynamic-PoR System

We mentioned earlier that we can plug any known construction of a dynamic-PoR system into the construction of Figure 6.6. To our knowledge, there are three known constructions of single-server dynamic PoR systems [23, 24, 75]. In this thesis, we instantiate Π with the construction given by Chandran *et al.* [24]. The reason why we picked the construction of Chandran *et al.* [24] in our instantiation is that it is based on codes of a specific form described below and its guarantees are explicitly stated using worst-case analysis. In contrast, the construction of Shi *et al.* [75] and that of Cash *et al.* [23] achieve only an amortized update guarantee. In the next paragraph, we discuss this issue with respect to the construction of Cash *et al.* [23].

The construction of Cash *et al.* [23] uses a cryptographic primitive known as the *oblivious* RAM [41] in a black-box manner. An *oblivious* RAM is a data structure that allows efficient read and write on the server without the server knowing which position has been read or written to or even whether it was a read or a write operation. The efficiency guarantees of the scheme of Cash *et al.* [23] depend on the efficiency guarantees of the underlying oblivious RAM. As a result, we get efficiency guarantees comparable to that of Chandran *et al.* [24] only when the efficiency is measured in an amortized sense, even with the most efficient known construction of oblivious RAM [56, 67, 78]. This is because the read and write efficiency of these known constructions of oblivious RAM are studied in the amortized sense. In the worst case, the update time of the construction of Cash *et al.* [23] has extra $\operatorname{poly}\log$ factors when instantiated with the state-of-the-art oblivious RAM of Shi *et al.* [74] that provides worst-case update guarantees. We note that the construction of Shi *et al.* [75] achieves a slightly better amortized update efficiency than Cash *et al.* [23] because it does not use an oblivious RAM in a black-box manner. However, they use a data structure to store the file which is structurally similar to that of an oblivious RAM, which results in an amortized update efficiency guarantee.

We now explore the construction of Chandran *et al.* [24] in more detail. Chandran *et al.* [24] first defined and constructed a new type of error-correcting code called a *locally* *updatable* code, and used it to construct their dynamic-PoR system. Intuitively, in a locally updatable code, one can update the codeword corresponding to a message m to a codeword corresponding to a message $m' = m + \xi$ by changing a small number of entries of the codeword, assuming ξ has Hamming weight one. They showed that, using an appropriate data structure, one can construct a locally updatable code using a locally decodable code (see Definition 2.3). We give a brief overview of their construction.

Recall the definition of a locally decodable code in Definition 2.3. Let loc be a constant. Let $\mathsf{Dec}_{\mathsf{LDC}}$ denote the decoding algorithm and $\mathsf{Enc}_{\mathsf{LDC}}$ denote the encoding algorithm of an $[N, k, \mathsf{loc}]_q$-LDC. In this section, we will always assume that $q = 2$. Therefore, we drop the subscript q and denote a locally decodable code by $[N, k, \mathsf{loc}]$-LDC.

We now describe the locally updatable and locally decodable codes of Chandran *et al.* [24]. We denote it by $\mathsf{LULDC} := (\mathsf{Enc}_{\mathsf{LULDC}}, \mathsf{Dec}_{\mathsf{LULDC}}, \mathsf{Update})$ and its parameters by $(N, k, \mathsf{loc}_r, \mathsf{loc}_u)$. The function $\mathsf{Enc}_{\mathsf{LULDC}} : (\mathbb{F}_q)^k \to (\mathbb{F}_q)^N$ is the encoding function that encodes a k-block message to an N-block message such that computing any i-th block of the message requires reading at most loc_r blocks of the codeword and updating any i-th block of message requires at most loc_u blocks of the codeword to be updated.

We first describe the algorithm that implements the encoding function. Let the message to be encoded using LULDC be k blocks long. Let a be an integer such that $2^a \leq k \leq 2^{a+1}$. The encoding algorithm picks an $[\widetilde{N}, k, \mathsf{loc}]$-$\mathsf{LDC}$ and a family of $[N_i, k_i, \mathsf{loc}_i]$-$\mathsf{LDC}$ such that $k_i = 2^i(\log k)$ and loc_i is constant for all $0 \leq i \leq a$. The block-length of the codeword is $N = \widetilde{N} + \sum_i N_i$.

The encoding algorithm uses a certain data structure. At a high level, the data structure has $a+1$ buffers, denoted by $\mathsf{buff}_0, \ldots, \mathsf{buff}_a$, such that buff_i can store N_i blocks. Initially, all blocks in $\mathsf{buff}_1, \ldots, \mathsf{buff}_a$ are set to a special symbol $\top$. We call any buffer that consists of only $\top$ an *empty* buffer. Additionally there is a special buffer, say buff, that can store up to $\widetilde{N}$ message blocks. The algorithm for $\mathsf{Enc}_{\mathsf{LULDC}}$ takes as input a message of block length k. To encode a message m of length k, we use the encoding function of $[\widetilde{N}, k, \mathsf{loc}]$-$\mathsf{LDC}$ and store the encoded message in buff. The contents of the buffers $\mathsf{buff}_0, \ldots, \mathsf{buff}_a, \mathsf{buff}$ at any point of time forms the codeword corresponding to the message m at that time.

Whenever we wish to update a block of the message, say the j-th block, we find the first empty buffer. Let the first empty buffer be buff_i. We decode $\mathsf{buff}_0, \ldots, \mathsf{buff}_{i-1}$ to get a vector of message blocks. We then include the new message block in this set of message blocks and encode the updated message using $[N_i, k_i, \mathsf{loc}_i]$-$\mathsf{LDC}$ and store it in buff_i. If no buffer is empty, then Enc decodes $\mathsf{buff}_0, \ldots, \mathsf{buff}_a$ and i is set to be $a+1$, place the new message block at appropriate block position and encode the new message with $[\widetilde{N}, k, \mathsf{loc}]$-$\mathsf{LDC}$. The buffers $\mathsf{buff}_0, \ldots, \mathsf{buff}_{i-1}$ are then set to be empty. Chandran *et* *al.* [24] showed that this is a locally updatable code.

Turning LULDC into a dynamic-PoR scheme. Using their construction of locally updatable and locally decodable codes, Chandran *et al.* [24] gave a construction of dynamic-PoR in the computational setting. They assume that it is computationally infeasible for adversarial provers to break a message authentication code (MAC) of constant output size. In their scheme, the $\mathsf{Verifier}$ computes the encoded message using $\mathsf{Enc}_{\mathsf{LULDC}}$ on the message and a corresponding authentication tag of the message. They used the code of Spielman [77] to construct the LULDC code. This is an error-correcting code that requires linear time to encode and decode a message. They also use a message authentication code, MAC, whose output is of constant size. To store a message m on a prover, they compute $M = \mathsf{Enc}_{\mathsf{LULDC}}(m)$ using the encoding function defined above. They also compute $\sigma[i] = \mathsf{MAC}(M[i])$. The $\mathsf{Verifier}$ stores $(\{M[1], \sigma[1]\}, \dots, \{M[n], \sigma[n]\})$ in buff and the rest of the buffers are set to $\top$ (here, $n = \widetilde{N}$, where $\widetilde{N}$ is as defined in the construction of LULDC). The resulting scheme is secure when the adversary is computationally bounded (this results in the use of security parameter in κ in the result stated below). To perform any update, the $\mathsf{Verifier}$ uses the update algorithm as presented above in the case of LULDC. Chandran *et al.* [24] used the extractor of Dodis, Vadhan, and Wichs [33, Section 4][3] to show the following.

Theorem 6.5. (Chandran *et al.* [24]) *Let $M = e(m) = M[1], \dots, M[n]$ be an encoding of an n-bit message m defined above such that each $M[i] \in \mathbb{F}_2$. Let κ be the security parameter. Then there exists a construction of a dynamic-PoR whose extractor runs in expected polynomial time. The dynamic-PoR requires a prover to store $O(n)$ field elements and the $\mathsf{Verifier}$ to store $O(\kappa)$ field elements. Also, a single audit can be done by accessing at most $O(\kappa \log n)$ blocks of the encoded message and an update can be performed in $O(\log n)$ time.*

6.4.3 Choosing an Appropriate Linear Secret Sharing Scheme

The scheme in Figure 6.6 uses a linear code and implicitly assumes that all the codewords in the linear code are available. This leads to subtle problems, which we illustrate through the means of the following example.

[3] Dodis, Vadhan, and Wichs [33] showed that if the underlying code used to store the message is a locally decodable code such that decoding any bits fails with at most negligible probability, then there exists an extractor that outputs the message with all but negligible probability.

Recall the generator matrix in Example 9,

$$\mathbf{G} := \begin{pmatrix} 1 & 0 & 0 & 9 & 2 & 11 & 7 \\ 0 & 1 & 0 & 4 & 11 & 12 & 11 \\ 0 & 0 & 1 & 8 & 9 & 4 & 11 \end{pmatrix}.$$

$\mathbf{G}$ is a generator matrix of a $[3, 7]_{13}$-linear code. Let $m = \begin{pmatrix}10, & 3, & 5\end{pmatrix} \in (\mathbb{F}_{13})^3$ be the message to be stored on the server. Therefore, $n = 1$ and $\rho = 4$. Then $m\mathbf{G} = \begin{pmatrix}10, & 3, & 5, & 12, & 7, & 10, & 2\end{pmatrix}$. The $\mathsf{Verifier}$ stores 12 on Prover_1, 7 on Prover_2, 10 on Prover_3, and 2 on Prover_4. During an update, say to m_2 from 3 to 4, the $\mathsf{Verifier}$ computes $\mathbf{e}_2^{\mathsf{T}}\mathbf{G}$ to get $\begin{pmatrix}0, & 1, & 0, & 4, & 11, & 12, & 11\end{pmatrix}$ and updates each prover using a single-server dynamic-PoR scheme to store the shares of the new message, which are $\begin{pmatrix}3, & 5, & 9, & 0\end{pmatrix}$.

Note that all the servers need to be updated for a single update in the above example. This is not an artifact of our example; in fact, most updates would require many servers to be updated. On the other hand, for efficiency reasons, it is desirable that, for a single update to the message, both the number of provers we need to update and the number of blocks on every prover that need to be updated should be small. Recall that the efficiency and storage guarantees stated in Theorem 6.4 depend on the row sparsity of the generator matrix of the linear code used to construct the LSSS. To achieve this, we use a linear code arising from a *low-density generator matrix* (LDGM). This gives a linear secret sharing scheme with a good privacy guarantee and efficient update time. Therefore, at the first glance, it seems that any low-density generator matrix suffices for our purpose. However, we have to be careful which low-density generator matrix we use to construct the LSSS.

To understand the complications, we first recall the construction of a ramp scheme from a linear code described in Section 2.4. Let C be a linear code of length $N = \rho + \mathsf{s}$ defined over a finite field $\mathbb{F}_q$. Let $(m_1, \ldots, m_{\mathsf{s}}) \in (\mathbb{F}_q)^{\mathsf{s}}$ be the message to be stored. The shares of a ramp scheme, described in Section 2.4, are computed as follows: select a random codeword of the form $(\mathbf{c}_1 = m_1, \ldots, \mathbf{c}_{\mathsf{s}} = m_{\mathsf{s}}, \mathbf{c}_{\mathsf{s}+1}, \ldots, \mathbf{c}_{\rho+\mathsf{s}}) \in C$, and define the shares as $(\mathbf{c}_{\mathsf{s}+1}, \ldots, \mathbf{c}_{\rho+\mathsf{s}})$. The result is a $(\mathsf{d}^{\perp} - \mathsf{s} - 1, N - \mathsf{d} + 1, N)$-linear secret sharing scheme, where $\mathsf{d}^{\perp}$ is the dual distance of C. In other words, we need to know (a bound on) the distance and dual distance of the underlying code. We revisit the known constructions of low-density generator matrices and low-density parity check matrices.

We start with the probabilistic construction of Bennatan and Burshtein [13] of low-density parity-check matrices. Recall from Theorem 2.2 that, if we use the codes whose parity check matrix is the one defined by Bennatan and Burshtein [13], with high probability, one can afford security against a constant fraction of the malicious servers. This is because the relative distance of their code is a constant and the security of the constructed

MPoR depends on the distance of the code. However, it is not clear what the dual distance is of this code. Moreover, the distance guarantee is true only with high probability (and not with probability 1). For these reasons, we look at alternate explicit constructions.

For example, we can use the explicit combinatorial constructions based on the Euclidean geometry or the one based on difference families by Johnson and Weller [50], described in Section 2.2.2 and Section 2.2.3, respectively.

We now look at the instantiation of dynamic- MPoR using the two explicit constructions mentioned above. We start with the construction based on the Euclidean geometry. If we instantiate the Dynamic-MPoR with a low-density parity check code based on the Euclidean geometry, then we need to determine u and w so that $\rho = 2^{uw} - \mathsf{s}$. In that case, using Theorem 2.3 and Theorem 2.6, we have

$$\tau_1 = \mathsf{d}^{\perp} - \mathsf{s} - 1 = (2^w + 2)2^{w(u-2)} - (\mathsf{s} + 1), \tag{6.1}$$

and

$$\tau_2 = \rho + \mathsf{s} - \mathsf{d} + 1 = 2^{uw} - \frac{2^{uw} - 1}{2^w - 1}. \tag{6.2}$$

If we use a 2-dimensional Euclidean geometry (so $u = 2$), then we have the following instantiation.

Example 20. Recall from Theorem 2.6 that if the dual distance of a code is $\mathsf{d}^{\perp}$, then we can take $1 \leq \mathsf{s} < \mathsf{d}^{\perp} - 2$ in Theorem 2.6. We set $\mathsf{s} = 1$ and $w = (\log_2(\rho + 1))/2$. Using Equation (6.1) and Equation (6.2), we have $\tau_1 = \sqrt{\rho + 1}$ and $\tau_2 = \rho - \sqrt{\rho + 1}$. Suppose the underlying single server dynamic-PoR scheme is an $(\eta, \nu/\tau_2, 1, 1)$-threshold secure MPoR, then substituting the values of τ_1 and τ_2 in Theorem 6.4, we get an $(\eta, \nu, \rho - \sqrt{\rho + 1}, \rho)$-threshold secure and $\sqrt{\rho + 1}$-private Dynamic-MPoR.

Note that we did not say anything about the update time efficiency of the MPoR in Example 20. This is because the above construction might not yield an efficient dynamic-MPoR and an update might require updating many servers. This is due to the fact that updating the provers requires computing the encoding of a basis element. This requires the use of a generator matrix of the code. Unfortunately, the generator matrix corresponding to the parity-check code matrix based on Euclidean geometry might be very dense, and, therefore, the number of servers that needs to be updated might be very large. This results in the same issue mentioned at the start of Section 6.4.

Fortunately, an easy remedy to the above problem exists if we have a linear code with a low-density generator matrix in standard form. Johnson and Weller [50] gave one such construction, which we described in detail in Section 2.2.3.

6.4.4 Putting Everything Together

We give an example by instantiating the scheme presented in Figure 6.6 with the LSSS of Johnson and Weller [50] and the single-server dynamic-PoR scheme of Chandran *et al.* [24]. We have all the required efficiency parameters of a single-server dynamic-PoR scheme; i.e., audit time, update time, and storage requirement of the scheme of Chandran *et al.* [24] through Theorem 6.5. Therefore, to instantiate Theorem 6.4 with a specific system, all that we need are the various parameters of the LSSS.

Let us first recount all the parameters of the LSSS that we need. We require the values of τ_1 and τ_2, which depend on the linear code on which the LSSS is based, and a bound on row sparsity of the underlying linear code. If we use a $(v, 3, 1)$-difference family whose size is L, then Theorem 2.4 gives the bound on the row sparsity of the linear code constructed by Johnson and Weller [50], i.e., $\zeta = 3(\mathsf{L}-1)+1$. Moreover, if we set $k = v(\mathsf{L}-1)$, $N = v\mathsf{L}$, and $\mathsf{s} = v$, then Lemma 2.5 gives the guarantee on the distance $\mathsf{d} = 3\mathsf{L}$ and dual distance $\mathsf{d}^{\perp} = 5$, which in turn defines τ_1 and τ_2 using Theorem 2.6.

Therefore, plugging Theorem 2.4, Lemma 2.5, and Theorem 6.5 into Theorem 6.4 gives us the following corollary.

Corollary 6.6. *Let* $v \equiv 1 \mod 6$ *be a prime number and* L *be as defined above. Let the message to be stored on the servers be of length* $\mathsf{s}n$ *bits long, such that* $\mathsf{s} = v \leq v(\mathsf{L}-1)$*. Set* $\rho = v\mathsf{L} - \mathsf{s} = v(\mathsf{L}-1)$*. Then there is an explicit construction of an* $(\eta, \nu, v(\mathsf{L}-1) - 2\mathsf{L} + 2, v(\mathsf{L}-1))$*-threshold secure and* 3*-private* Dynamic-MPoR*, where every prover has to store* $O(n)$ *bits. Any audit takes* $O(\kappa \mathsf{L} \log n)$ *time and an update takes* $O(\mathsf{L} \log n)$ *time.*

In the above corollary, we assumed that the message to be stored is $\mathsf{s}n$ bits in length. This is without any loss of generality because we can always pad the message appropriately to make it a multiple of s bits in length.

6.5 Optimization Using the Shacham-Waters Scheme

In the last three sections, we gave constructions of MPoR schemes using ramp schemes, linear secret-sharing schemes, replication codes, and a single-prover-PoR system. In this section, we show a specific instantiation of our scheme using the keyed scheme of Shacham and Waters [66, 72] for a single server PoR system. Note that this does not provide a dynamic-MPoR scheme.

Recall from Section 4.4 that, in the setting of unconditional security, any keyed PoR scheme is considered to be secure when the success probability of the prover $\mathcal{P}$, denoted by $\mathsf{succ}(\mathcal{P})$, is the the average success probability of the prover over all possible keys (Theorem 4.13). The same reasoning extends to MPoR systems. Therefore, in what follows, when we say a scheme is an (η, ν, τ, ρ)-threshold secure scheme, the term η is the average success probability where the average is over all possible keys. We use $\mathsf{succ}_{\mathsf{avg}}(\mathcal{P})$ to denote the average success probability of a prover $\mathcal{P}$ over all possible keys.

We recall the parameters in Equation 4.6 (see Theorem 4.13). Let ℓ be the hamming weight of the challenges made by the $\mathsf{Verifier}$ and d be the Hamming distance of the message space. We denote $\eta = 1 - \frac{\widetilde{\mathsf{d}}(q-1)}{2\gamma q}$, where q is the size of the underlying field, $\gamma = q^n$, and

$$\widetilde{\mathsf{d}} \approx \binom{n}{\ell}(q-1)^{\ell} - \binom{n-\mathsf{d}}{\ell}(q-1)^{\ell} - \sum_{w \geq 1} \binom{\mathsf{d}}{w}\binom{n-\mathsf{d}}{\ell - w}\frac{(q-1)^{\ell}}{q} \tag{6.3}$$

is the approximate value of the Hamming distance of the response code for $\mathsf{Modified\ Shacham\text{-}Waters\ scheme}$.

6.5.1 Extension of the Keyed Shacham-Waters Scheme to MPoR

If we instantiate the $\mathsf{Rep\text{-}MPoR}$ scheme (described in Section 6.2) with the Shacham-Waters scheme (described in Figure 4.4), then we need one key that consists of $n+1$ values in $\mathbb{F}_q$. However, in this case, we do not have any privacy. In particular, we have the following extension of Corollary 6.2.

Corollary 6.7. *Let Π be an $(\eta, 0, 0, 1)$-PoR system, described in Figure 4.4, with a response code of Hamming distance $\widetilde{\mathsf{d}}$ and the size of challenge space γ (where $\widetilde{\mathsf{d}}$ is approximated by Equation (6.3)). Then $\mathsf{Rep\text{-}MPoR}$, instantiated with the Shacham-Waters scheme of Figure 4.4, is an MPoR system with the following properties:*

1. *Privacy: It is 0-private.*
2. *Security: It is $(\eta, 0, 1, \rho)$-threshold secure, where $\eta = 1 - \frac{\widetilde{\mathsf{d}}(q-1)}{2\gamma q}$.*
3. *Storage Overhead: $\mathsf{Verifier}$ needs to store $n+1$ field elements and every Prover_i needs to store $2n$ field elements.*

Proof. The results follow by combining Theorem 4.13 with Corollary 6.2. □

The issue with the Rep-MPoR scheme is that there is no confidentiality of the file. In what follows, we improve the privacy guarantee of the MPoR scheme described above. Our starting point would be an instantiation of the Ramp-MPoR scheme, defined in Figure 6.2, with the Shacham-Waters scheme. We then reduce the storage on the Verifier through a series of steps.

6.5.2 Optimized Version of the Multi-server Shacham-Waters Scheme

We follow two steps to get a MPoR scheme based on the Shacham-Waters scheme with a reduced storage requirement for the Verifier, while improving the confidentiality guarantee.

1. In the first step, stated in Theorem 6.8, we improve the privacy guarantee of the MPoR scheme to get a τ_1-private MPoR scheme (where $\tau_1 < \rho$ is an integer). The Verifier in this scheme has to store $\rho(n+1)$ field elements. When the underlying field is $\mathbb{F}_q$, the verifier has to store $\rho(n+1)\log q$ bits.

2. In the second step, stated in Theorem 6.9, we reduce the storage requirement of the Verifier from $\rho(n+1)$ to $\tau_1(n+1)$ field elements for some integer $\tau_1 < \rho$ without affecting the privacy guarantee. When the underlying field is $\mathbb{F}_q$, the verifier has to store $\tau_1(n+1)\log q$ bits.

Step 1. To improve the privacy guarantee of Corollary 6.7 to say, τ_1-private (as per Definition 6.3), we use a Ramp-MPoR scheme and ρ different keys, where each key consists of $n+1$ values in $\mathbb{F}_q$. The Verifier generates ρ shares of every message block using a ramp scheme, then encodes the shares, and finally computes the tag for each of these encoded shares (see Figure 4.4).

We follow with more details. Let $m = (m[1], \dots, m[k])$ be the message. The Verifier computes the shares of every message block $(m[1], \dots, m[k])$ using a (τ_1, τ_2, ρ)-Ramp scheme. It then encodes all the shares using the encoding scheme of the PoR scheme. Let the resulting encoded shares be $M_i[1], \dots, M_i[n]$ for $1 \leq i \leq \rho$. In other words, the result of the above two steps are ρ encoded shares, each of which is an n-tuple in $(\mathbb{F}_q)^n$. The Verifier now picks random values $a^{(i)}, b_1^{(i)}, \dots, b_n^{(i)} \in \mathbb{F}_q$ for $1 \leq i \leq \rho$ and computes the tags as in Figure 4.4, i.e.,

$$S_i[j] = b_j^{(i)} + a^{(i)} M_i[j] \qquad \text{for } 1 \leq i \leq \rho, 1 \leq j \leq n.$$

The verifier gives Prover_i the tuple of encoded message $(M_i[1], \ldots, M_i[n])$ and the corresponding tags $(S_i[1], \ldots, S_i[n])$. We call this scheme Basic-MPoR scheme. The following is straightforward from Theorem 6.1.

Theorem 6.8. *Let* Π *be an* $(\eta, 0, 0, 1)$-PoR *scheme, described in Figure 4.4, with a response code of Hamming distance* $\widetilde{\mathsf{d}}$ *and the size of challenge space* $\gamma = q^n$ *(where* $\widetilde{\mathsf{d}}$ *is given by Equation (6.3)). Let* Ramp *be a* (τ_1, τ_2, ρ)*-ramp scheme. Then* Basic-MPoR *defined as above is an* MPoR *scheme with the following properties:*

1. *Privacy:* Basic-MPoR *is* τ_1*-private.*

2. *Security:* Basic-MPoR *is* $(\eta, 0, \tau_2, \rho)$*-threshold secure, where* $\eta = 1 - \frac{\widetilde{\mathsf{d}}(q-1)}{2\gamma q}$.

3. *Storage Overhead: The* Verifier *needs to store* $\rho(n+1)$ *field elements and every* Prover_i *needs to store* $2n$ *field elements.*

In the construction mentioned above, the Verifier needs to store $\rho(n+1)$ elements of $\mathbb{F}_q$, which is almost the same as the total storage requirements of all the provers. We encountered the same issue in the single-server setting as well (see Section 4.4), where the Verifier has to store as much secret information as the size of the message. This seems to be the general drawback in the unconditional secure setting. However, in the case of MPoR, we can improve the storage requirement of the Verifier as shown in the next two steps.

Step 2. In this step, we improve the above-described MPoR scheme to achieve considerable reduction on the storage requirement of the Verifier. The resulting scheme also provides unbounded audit capability against computationally unbounded adversarial provers, and it also ensures τ_1-privacy.

The main observation that results in the reduction in the storage requirements of the Verifier is the fact that we can partially derandomize the keys generated by the Verifier. We use one of the most common techniques in derandomization. The keys in this scheme are generated by τ_1-wise independent functions.[4] Our construction works as follows: we pick $n+1$ random polynomials, $f_1(x), \ldots, f_n(x), g(x) \in \mathbb{F}_q[x]$, each of degree at most $\tau_1 - 1$. Then the Verifier computes the secret key by evaluating the polynomials $f_j(x)$ and $g(x)$ on ρ different values, say

$$b_j^{(i)} = f_j(i) \qquad \text{and} \qquad a_i = g(i)$$

[4] A function is a τ_1*-wise independent* function if every subset of τ_1 outputs is independent and equally likely. It should be noted that this does not imply that all the outputs of the function are mutually independent.

for $1 \leq j \leq n$, and $1 \leq i \leq \rho$. The $\mathsf{Verifier}$ then computes the encoded shares and their corresponding tags as in $\mathsf{Basic\text{-}MPoR}$, i.e.,

$$S_i[j] = b_j^{(i)} + a^{(i)} M_i[j] \qquad \text{for } 1 \leq i \leq \rho, 1 \leq j \leq n.$$

Figure 6.7 is the formal description of this scheme. For the scheme described in Figure 6.7, we prove the following result.

Theorem 6.9. *Let* $\mathsf{Ramp} = (\mathsf{ShareGen}, \ \mathsf{Reconstruct})$ *be a* (τ_1, τ_2, ρ)*-ramp scheme. Let* Π *be a single-prover Shacham-Water scheme, described in Figure 4.4, with a response code of Hamming distance* $\widetilde{\mathsf{d}}$ *and the size of challenge space* γ*. Then* $\mathsf{SW\text{-}MPoR}$*, defined in Figure 6.7, is an* MPoR *system with the following properties:*

1. *Privacy:* $\mathsf{SW\text{-}MPoR}$ *is* τ_1*-private.*
2. *Security:* $\mathsf{SW\text{-}MPoR}$ *is* $(\eta, 0, \tau_2, \rho)$*-threshold secure, where* $\eta = 1 - \frac{\widetilde{\mathsf{d}}(q-1)}{2\gamma q}$.
3. *Storage Overhead:* $\mathsf{Verifier}$ *needs to store* $\tau_1(n+1)$ *field elements and every* Prover_i *(for* $1 \leq i \leq \rho$*) needs to store* $2n$ *field elements.*

Proof. The privacy guarantee of $\mathsf{SW\text{-}MPoR}$ is straightforward from the secrecy property of the underlying ramp scheme.

For the security guarantee, we have to show an explicit construction of $\mathsf{Extractor}$, that on input proving algorithms $\mathcal{P}_1, \ldots, \mathcal{P}_\rho$, outputs m if $\mathsf{succ}(\mathcal{P}_i) > \eta$ for at least τ_2 proving algorithms. However, there is a subtle issue that we have to deal with before using the proof of Theorem 6.1, because of the relation between every message and tag pair. Recall from Section 4.4 that, if the adversarial prover learns the secret key, then it can break the PoR scheme. We first argue that a set of τ_1 colluding provers cannot have an undue advantage from exploiting the linear structure of the message-tag pairs.

We now prove that any set of τ_1 provers do not learn anything about the keys generated using $n+1$ polynomials of degree at most $\tau_1 - 1$. The idea is very similar to the single-prover case. Recall from Section 4.4 that in the single prover case, for an n-tuple encoded message, the key is a tuple of $n+1$ uniformly random elements $(a, b_1, \ldots, b_n)$ in $\mathbb{F}_q$. Further, from the point of view of a prover, there are q possible keys — the value of a determines the n-tuple $(b_1, \ldots, b_n)$ uniquely, but a is completely undetermined (see Figure 3.2 for details). In the MPoR case, we have ρ keys. Each prover in a given set of τ_1 provers has q possible keys, as discussed above. However, it is conceivable that they can use their collective

Input: The Verifier gets a message $m = (m[1], \ldots, m[n])$ as input. Let $\mathsf{Prover}_1, \ldots, \mathsf{Prover}_\rho$ be the set of ρ provers. Let q be a prime number greater than ρ.

Initialization Stage: The Verifier performs the following steps

1. The Verifier choses $n+1$ random polynomials of degree at most $\tau_1 - 1$, $f_1(x), \ldots, f_n(x), g(x) \in \mathbb{F}_q[x]$ and a (τ_1, τ_2, ρ)-ramp scheme $\mathsf{Ramp} = (\mathsf{ShareGen}, \mathsf{Reconstruct})$.
2. For every server i, the Verifier does the following:
   (a) Compute ρ shares of every message block using the share generation algorithm of Ramp as follows: $(m_1[j], \ldots, m_\rho[j]) \leftarrow \mathsf{ShareGen}(m[j])$ for $1 \leq j \leq n$.
   (b) The Verifier encodes the message as $e(m_i[j]) = M_i[j]$ for $1 \leq j \leq n, 1 \leq i \leq \rho$.
   (c) Compute $b_1^{(i)} = f_1(i), \ldots, b_n^{(i)} = f_n(i), a^{(i)} = g(i)$.
   (d) Compute the tag $S_i[j] = b_j^{(i)} + a^{(i)} M_i[j]$ for $1 \leq j \leq n$.
3. The Verifier gives $\{(M_i[j], S_i[j])\}_{1 \leq j \leq n}$ to Prover_i.

Challenge Phase: During the audit phase, Verifier picks a prover, Prover_i, and runs the challenge-response algorithm of a single-server Shacham-Waters scheme. It computes the corresponding keys by computing the random polynomials chosen during the set up phase.

Figure 6.7: MPoR Using Optimized Shacham-Waters scheme (SW-MPoR).

knowledge to learn something about the keys. In what follows, we show that they cannot determine any additional information about their keys by combining the information they hold collectively.

Let $I = \{i_1, \ldots, i_{\tau_1}\}$ be the indices of any arbitrary set of τ_1 provers. Let S_i denote the set of possible keys for Prover_i, for $i \in I$. Consider any list of τ_1 keys $(K_{i_1}, K_{i_2}, \ldots, K_{i_{\tau_1}})$. Recall that K_i (for $i \in I$) has the form $\left(a^{(i)}, b_1^{(i)}, \ldots, b_n^{(i)}\right)$, where $a^{(i)}$ and $b_j^{(i)}$ (for $1 \leq j \leq n$) are generated by random polynomials of degree at most $\tau_1 - 1$. We first consider $a^{(i)}$ (for $i \in I$). Note that the vector $\left(b_1^{(i)}, \ldots, b_n^{(i)}\right)$ is defined uniquely by $a^{(i)}$ and the set of all encoded message-tag pairs. We have already shown that any set of τ_1 provers cannot learn anything about the random polynomial $g(x)$ used to generate the $a^{(i)}$ for all $i \in I$. We use the following well known fact to show the any set of τ_1 provers do not learn any additional information about the keys.

Fact 6.10. *Let t be a positive integer, let q be a prime number, and let $\mathbb{F}_q$ be a finite field. Let $h_0, h_1, \ldots, h_{t-1} \in \mathbb{F}_q$ be random elements picked uniformly at random. Define $h(x) = \sum_{i=0}^{t-1} h_i x^i$ for all $\alpha \in \mathbb{F}_q$. Then,*

$$\mathsf{Pr}\left[h\left(x_1\right) = y_1 \wedge \ldots \wedge h\left(x_\tau\right) = y_t\right] = \prod_{i=1}^{t} \mathsf{Pr}\left[h(x_{\alpha_i}) = y_i\right]. \tag{6.4}$$

Since $h(x)$ is uniformly distributed in $\mathbb{F}_q$, the probability computed in Equation (6.4) is actually equal to q^{-t}.

By construction, $g(x)$ is a random polynomial of degree at most $\tau_1 - 1$. Fact 6.10 then implies that any combination of $\left\{a^{(i)}\right\}_{i \in I}$ is equally likely. A similar argument, with the $a^{(i)}$'s replaced by the $b_j^{(i)}$'s (for all $i \in I$ and $1 \leq j \leq n$) and the polynomial $g(x)$ replaced by $f_j(x)$ (for $1 \leq j \leq n$), gives that all set of τ_1 keys are equally likely. In other words, the set of provers in the set I cannot determine any additional information about their keys by combining the information they hold collectively.

We now complete the security proof by describing an Extractor that outputs the file if τ_2 provers succeeds with high enough probability. The description of the Extractor and its analysis is same as that of Theorem 6.1. We give it for the sake of completeness.

1. Extractor chooses τ_2 provers and runs the extraction algorithm of the underlying single-server PoR system on each of these provers. In the end, it outputs $\widehat{M}_{i_j}$ for the corresponding provers Prover$_{i_j}$. It defines $\mathcal{S} \leftarrow \{\widehat{m}_{i_1}, \ldots, \widehat{m}_{i_{\tau_2}}\}$. Note that the Extractor of the underlying PoR scheme has already computed e^{-1} on the set $\left\{\widehat{M}_{i_1}, \ldots, \widehat{M}_{i_{\tau_2}}\right\}$ as outputted in Step 2 of the extractor in the proof of Theorem 4.13.
2. Extractor invokes the Reconstruct algorithm of the underlying ramp scheme with the elements of $\widetilde{\mathcal{S}}$ to compute m'.

Now note that the Verifier interacts with every Prover$_i$ independently. We know from the security of the underlying PoR scheme of Shacham-Waters (Theorem 4.13) that there is an extractor that always outputs the encoded message whenever $\mathsf{succ}_{\mathsf{avg}}(\mathcal{P}_i) \geq \eta$. Therefore, if all the τ_2 chosen proving algorithms succeed with probability at least η over all possible keys, then the set $\mathcal{S}$ will have τ_2 correct shares. From the correctness of the Reconstruct algorithm and $e^{-1}(\cdot)$, we know that the message output in the end by the Extractor will be the message m.

For the storage requirement, the Verifier has to store the coefficients of all the random polynomials $f_1(x), \ldots, f_n(x), g(x)$, which amounts to a total of $\tau_1(n+1) = \tau_1 n + n$ field elements. $\square$

6.6 Conclusion

In this chapter, we studied PoR systems when multiple provers are involved (MPoR). We motivated and defined the security of MPoR in the worst-case (Definition 6.1) and the average-case (Definition 6.2) setting, and extended the hypothesis testing techniques stated in Chapter 4 to the multi-server setting. We also motivated the study of confidentiality of the outsourced message. We gave MPoR schemes which are secure under both these security definitions and provide reasonable confidentiality guarantees even when there is no restriction on the computational power of the servers. We also gave an efficient dynamic-MPoR system in the computational setting. Our construction performs a generic transformation of a single-server dynamic-PoR system to a dynamic-MPoR system. We also looked at one specific instantiation of our construction which provides efficient updates and audits while giving reasonable security and privacy guarantee. In the end of this chapter, we looked at an optimized version of MPoR system when instantiated with the unconditionally secure version of the Shacham-Waters scheme studied in Chapter 4. We exhibited that, in the multi-server setting with computationally unbounded provers, one can overcome the limitation that the verifier needs to store as much secret information as the provers.

Chapter 7

Conclusion and Future Work

The major focus of this thesis has been the integrity of data stored on a cloud. We studied various problems in a well-known security model without making any assumptions on the computational power or the resources of a server. In Section 7.1, we first reiterate our contributions with respect to the questions we raised in Chapter 1. We then state in Section 7.2 some of the open problems that we believe are important to understand various privacy and integrity issues of large data.

7.1 Contributions of this Thesis

In this thesis, we studied the well-established notion of *proof-of-retrievability* systems in an unconditional security framework in single and multiple server settings. We also revisited the definition that captures the notion of *data-possession* and showed that a stronger guarantee can be fulfilled that better instantiates this notion.

In summary, this thesis provides answers to the questions mentioned in Chapter 1. For the benefit of the reader, we review both the questions and the answers below.

Question 1: Is it possible to reduce the dependence of the security of PoR on some underlying hardness assumption or existence of secure cryptographic primitives? In Chapter 4, we showed the exact criterion under which we can always extract a file if a computationally unbounded server acts maliciously. In addition to this, we also showed how classical statistical techniques can be used to evaluate whether the responses of the prover are accurate enough to permit successful extraction.

Question 2: What is a natural definition that captures the notion that the server possesses the file when it responds to the audits, and are there protocols that satisfy this definition? To answer this question, in Chapter 5, we gave a new definition that tries to capture the requirement that the server must have an actual copy of the data in its memory space while it executes the challenge-response protocol. To answer the second question, we gave some example protocols achieving this goal in the random oracle model.

Question 3: What is a reasonable definition of multi-server secure cloud storage systems that reflects the natural integrity requirement of the data? Can we construct a cloud storage system that is secure under this definition? We answered this question in Chapter 6. We gave two definitions of security in the multiple server setting by modelling successful extraction of the file when a threshold of servers have high enough success probability and as well as when the average of the success probabilities of all the servers is high enough. We presented some protocols that satisfy these definitions. We also studied efficient dynamic updates when a file is stored on multiple servers and we gave an efficient protocol when the servers are computationally bounded. We also described an optimized version of a multi-server Shacham-Waters scheme [72] when the servers are computationally unbounded.

7.2 Open Problems

This thesis leaves open problems pertaining to large data, which might be of independent interest. We do not claim the list to be in any way exhaustive, but we believe these are important questions that need to be answered to improve our understanding of security in cloud storage. We also point out some of the related works that might be useful in solving these questions.

7.2.1 Kolmogorov Complexity and Secure Cloud Storage

The general bound proven in Section 5.3 depends on the assumption that the adversary is only permitted a fixed number of calls to the random oracle before it constructs the proving algorithm $\mathcal{P}$. An alternative approach would be to analyze the situation if the *memory* used by $\mathcal{P}$ is upper-bounded. However, it is possible that $\mathcal{P}$ could be constructed to compute certain responses $\mathsf{hash}(M \parallel c)$ without explicitly storing them. In order to accommodate this possibility, the analysis could be perhaps done in the setting of *Kolmogorov complexity* [57].

The theory of Kolmogorov complexity asserts that "almost all" strings of a given length cannot be generated more efficiently (i.e, using less space) than they can be stored. Because the outputs of a random oracle are completely random, it should be possible to obtain theorems about the success probability of proving algorithms having a specified Kolmogorov complexity [57].

Husain *et al.* [47] used Kolmogorov complexity in conjunction with list decoding to deal with the question of storage enforcement. They prove that the server must store the file in an invertible format M' such that the size of M' is at most an additive factor away from the Kolmogorov complexity of M. Their security reduction uses an extractor paradigm, and the security reduction is a non-uniform reduction; i.e., the reduction needs an advice string (see the book by Arora and Barak [2] for a detailed definition of non-uniform reduction). On the other hand, it is preferable to have a uniform reduction. Therefore, an important question is to give a uniform reduction to prove security for cloud storage while considering Kolmogorov complexity.

7.2.2 Game-theoretic View of Secure Cloud Storage

A rigorous study of security requirements of real-world cloud computing is extremely important. On the other hand, if we look at real-world scenarios, there seems to be a gap between the real-world scenario and the existing security definitions. For instance, the well-known security definitions of secure cloud storage assume that servers are malicious. In the real world, servers are more likely to be rational and directed towards incentives and reputation. Therefore, it might be more reasonable to view secure cloud storage from a game-theoretic point of view. One of the pressing issues here is that it is not understood how we can model the extractor paradigm in a game-theoretic setting. If not, is there another natural way to model security?

There have been some recent works that define and construct cryptographic primitives in the game-theoretic model [32, 53, 54]. These works culminated in the notion of rational cryptography. In traditional cryptography, the adversary is modelled as malicious entity whose goal is to disrupt the execution of a protocol. In the model of rational cryptography, we assume that the adversaries are "rational" and not malicious; i.e., we assign a utility function corresponding to every possible outcome of the protocol and the primarily goal of an adversary is to maximize its utility.

Many cryptographic primitives, such as secret sharing [46] and secure multiparty computation [1], have been studied in this model. There has been considerable work that revisits *interactive proofs* in the game-theoretic setting by assuming that the prover is motivated by the utility it gets from the execution of the protocol [8, 45]. It would be interesting to see if the techniques used in the literature of rational secret sharing and complexity theory could be used in providing secure cloud storage when the servers are rational.

7.2.3 Hourglass Scheme

In a recent work, van Dijk *et al.* [86] considered the scenario in which a client wants to store an encryption of its file, but does not have enough resources to perform the encryption on its own. The client trusts the server with the confidentiality of the file, but does not trust the server would actually perform the encryption (one possible reason could be the resource-intensive encryption algorithm). Therefore, the client provides the server with both the file and the key under which the file should be encrypted. The problem is to somehow force the server to store the encrypted file and not the plaintext file.

van Dijk *et al.* [86] formalized this problem and gave a scheme which they call an *hourglass* scheme. They rely on certain limitations of the server to prove the security of their system. However, they do not exactly solve the problem they described. More precisely, van Dijk *et al.* [86] motivate the problem that the server should store the file in an encrypted format; however, their solution forces the server to store the file in a new file format using an application of a specially designed function called an *hourglass function* such that it is easier to move from the new file format to the encrypted file format, but it is not easier to compute the new file format from the unencrypted file.

We believe that a more natural way to look at the problem motivated by van Dijk *et al.* [86] is to view it in a game-theoretic model [64]. In this model, we give a utility function to the server that depends on how many resources it needs to invest in order to encrypt the file and the incentive it gets from the client. The resources that the server employs might be time, power, and software, while the benefits it gets from the client for storing the correct format can be monetary. We believe this is a more practical setup and the question now is whether we can construct a system that forces the server to store the file in encrypted form. In other words, can we construct a system in which storing the encrypted version of the file is a *dominant strategy*; i.e., a strategy that gives the strictly highest utility.

7.2.4 Random Projection and Differential Privacy

The third potential application of cloud storage mentioned in Chapter 1 is to provide accurate answers on queries made by an *analyst* on the outsourced file held by a *curator*. The database could store many different types of information about an individual, some of which may be confidential. Differential privacy [34] guarantees that the privacy of every confidential entry in the database is maintained while answering the queries made by a malicious analyst on the whole database, even if the malicious analyst performs arbitrary post-processing on the answers it receives from the curator.

Until recently, algorithmic developments in differential privacy were focussed on computing various forms of statistical queries efficiently. However, with the recent deanonymization of the Netflix public data sets [63], there have been significant efforts to design algorithms for privacy-preserving algebraic tasks such as *low-rank approximation.* In the non-private setting, *low-dimension embeddings*, which project a set of points from a high-dimensional space to a lower-dimensional space, have been used to perform many tasks related to numerical linear algebra, including low-rank approximation, efficiently.

Extending the methods of Blocki *et al.* [17] and Upadhyay [83], we provide in a separate work [84] mechanisms that give positive results for private analogues of these algorithms using space-efficient data structures. This was further improved in our subsequent work [85] to get an update-time-efficient algorithm. These works open up a wide avenue of research.

One of the open problems is to perform efficient low-dimension embeddings while using linear random samples and preserving differential privacy. The result in Upadhyay [85] has a slackness in the terms of random samples. The main source of slackness is in proving one of the intermediate lemmata. If we can somehow improve this bound by some other technique that does not require a lot of random bits, it would lead to an improvement of the overall bound. One technique that is often useful in such a scenario is decoupling where, under limited dependence, we prove results equivalent to that of identically and independently distributed random variables. The excellent book by De la Pena and Gine [30] shows many such examples. It would be interesting if any such methods can be applied in our context.

7.2.5 Delegation of Property Testing

One of the potential applications of secure storage mentioned in Chapter 1 is to perform resource-heavy computations enabling a weak client to delegate such computations to the servers [39, 42]. One of the research problems is motivated from a potential application of delegation of computation in differential privacy. Succinctly, it can be described as follows. Is there an efficient method to delegate the testing of the property whether a function is 1-Lipschitz under a specified norm[1] over a local domain? Property testing is a decision problem in which we are given a function and its domain. The task is to find out whether the function has a particular property over the specified domain or whether it fails to have the property over an arbitrary large fraction of the domain. The restriction on the algorithm which solves this task is that it can only receive the output of the function on a small subset of the domain space.

Property testing has received wide attention over the last few years and efficient algorithms to test many properties of an n-bit input function with domain $\{0,1\}^n$ have been proposed [71]. In fact, testing whether a function is 1-Lipschitz is efficient on *hypergrids* [49]. However, the delegation version of this problem has not been studied that much, with the only exception being the work of Rothblum, Vadhan, and Widgerson [70].

This question is important because many results in differential privacy assume that the query function is 1-Lipschitz and privacy is guaranteed only if the query function is 1-Lipschitz. Therefore, it is important in practice to test this condition before answering the query. However, in differential privacy, the input to a query function is a database, and it is much more logical to assume that a database has entries from a local subset of all possible inputs. For instance, a database containing the height of adults is more likely to contain values in the range of 1.4 to 1.9 meters. It is an important question to perform property testing over a local domain in a more general class of functions.

[1] A function $f(\cdot)$ over domain $\mathcal{D}$ is c-Lipschitz if for all $x, y \in \mathcal{D}$, we have $\|f(x) - f(y)\|_N \leq c\|x - y\|_N$ for a specified norm N.

References

[1] Ittai Abraham, Danny Dolev, Rica Gonen, and Joe Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 53–62. ACM, 2006.

[2] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[3] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Transactions on Information System and Security*, 14(1):12, 2011.

[4] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn X. Song. Provable data possession at untrusted stores. In *ACM Conference on Computer and Communications Security*, pages 598–609, 2007.

[5] Giuseppe Ateniese, Özgür Dagdelen, Ivan Damgård, and Daniele Venturi. Entangled cloud storage. *IACR Cryptology ePrint Archive*, 2012:511, 2012.

[6] 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 Networks*, page 9. ACM, 2008.

[7] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *Advances in Cryptology–ASIACRYPT 2009*, pages 319–333. Springer, 2009.

[8] Pablo Daniel Azar and Silvio Micali. Rational proofs. In *Proceedings of the forty-fourth Annual ACM Symposium on Theory of Computing*, pages 1017–1028. ACM, 2012.

[9] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Advances in Cryptology—CRYPTO'92*, pages 390–420. Springer, 1993.

[10] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[11] Shai Ben-David, Benny Chor, and Oded Goldreich. On the theory of average case complexity. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, pages 204–216. ACM, 1989.

[12] Josh Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *Proceedings on Advances in Cryptology – CRYPTO 1990*, pages 27–35. Springer-Verlag New York, Inc., 1990.

[13] Amir Bennatan and David Burshtein. On the application of LDPC codes to arbitrary discrete-memoryless channels. *IEEE Transactions on Information Theory*, 50(3):417–438, 2004.

[14] Rajendra Bhatia. *Matrix Analysis*, volume 169. Springer Science & Business Media, 2013.

[15] George Robert Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, volume 48, pages 313–317, 1979.

[16] George Robert Blakley and Catherine Meadows. Security of ramp schemes. In *Advances in Cryptology – CRYPTO 1985*, pages 242–268. Springer, 1985.

[17] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. The johnson-lindenstrauss transform itself preserves differential privacy. In *Foundations of Computer Science*, pages 410–419, 2012.

[18] Andrej Bogdanov and Luca Trevisan. Average-case complexity. *arXiv preprint cs/0606037*, 2006.

[19] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, pages 43–54. ACM, 2009.

[20] Gilles Brassard, Sophie Laplante, Claude Crépeau, and Christian Léger. Computationally convincing proofs of knowledge. In *Symposium on Theoretical Aspects of Computer Science*, pages 251–262. Springer, 1991.

[21] Neil J Calkin, Jennifer D Key, and Marialuisa J De Resmini. Minimum weight and dimension formulas for some geometric codes. *Designs, Codes and Cryptography*, 17(1-3):105–120, 1999.

[22] George Casella and Roger L. Berger. *Statistical Inference*. Duxbury Pacific Grove, CA, 2002.

[23] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious RAM. In *Advances in Cryptology–EUROCRYPT 2013*, pages 279–295. Springer, 2013.

[24] Nishanth Chandran, Bhavana Kanukurthi, and Rafail Ostrovsky. Locally updatable and locally decodable codes. In *Theory of Cryptography*, pages 489–514, 2014.

[25] Charles J Colbourn. *CRC Handbook of Combinatorial Designs*. CRC press, 2010.

[26] Thomas M Cover and Joy A Thomas. *Elements of Information Theory*. John Wiley & Sons, 2012.

[27] Reza Curtmola, Osama Khan, Randal C. Burns, and Giuseppe Ateniese. MR-PDP: Multiple-replica provable data possession. In *International Conference on Distributed Computing Systems*, pages 411–420, 2008.

[28] Fergus Daly, David J. Hand, M. C. Jones, A. D. Lunn, and K. J. McConway. *Elements of Statistics*. Addison-Wesley Publishing Company, 1995.

[29] Matthew C Davey and David JC MacKay. Low density parity check codes over GF(q). In *Information Theory Workshop, 1998*, pages 70–71. IEEE, 1998.

[30] Victor De la Pena and Evarist Giné. *Decoupling: From Dependence to Independence*. Springer Science & Business Media, 1999.

[31] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.

[32] Yevgeniy Dodis, Shai Halevi, and Tal Rabin. A cryptographic solution to a game theoretic problem. In *Advances in Cryptology—Crypto 2000*, pages 112–130. Springer, 2000.

[33] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Theory of Cryptography*, pages 109–127, 2009.

[34] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, pages 265–284, 2006.

[35] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 213–222. Acm, 2009.

[36] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[37] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the twenty-second Annual ACM Symposium on Theory of Computing*, pages 416–426, 1990.

[38] Robert G Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1):21–28, 1962.

[39] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010*, pages 465–482. Springer, 2010.

[40] Oded Goldreich. Foundations of Cryptography (in two volumes: Basic Tools and Basic Applications), 2001.

[41] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[42] Shafi Goldwasser, Yael T. Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the fortieth Annual ACM Symposium on Theory of Computing*, pages 113–122. ACM, 2008.

[43] M. Gonzalez Lopez, F. J. Vazquez Araujo, L. Castedo, and J. Garcia Frias. Design of serially-concatenated low-density generator matrix codes using exit charts. In *6th International ITG-Conference on Source and Channel Coding (TURBOCODING)*, pages 1–6, April 2006.

[44] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

[45] Siyao Guo, Pavel Hubáček, Alon Rosen, and Margarita Vald. Rational arguments: single round delegation with sublinear verification. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, pages 523–540. ACM, 2014.

[46] Joseph Halpern and Vanessa Teague. Rational secret sharing and multiparty computation. In *Proceedings of the thirty-sixth Annual ACM symposium on Theory of computing*, pages 623–632. ACM, 2004.

[47] Mohammad I. Husain, Steve Ko, Atri Rudra, and Steve Uurtamo. Almost universal hash families are also storage enforcing. *arXiv preprint arXiv:1205.1462*, 2012.

[48] Russell Impagliazzo. A personal view of average-case complexity. In *Structure in Complexity Theory Conference, 1995., Proceedings of Tenth Annual IEEE*, pages 134–147. IEEE, 1995.

[49] Madhav Jha and Sofya Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. *SIAM Journal on Computing*, 42(2):700–731, 2013.

[50] Sarah J. Johnson and Steven R. Weller. A family of irregular LDPC codes with low encoding complexity. *IEEE Communications Letters*, 7(2):79–81, 2003.

[51] Ari Juels and Burton S Kaliski Jr. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 584–597. ACM, 2007.

[52] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In *Financial Cryptography and Data Security*, pages 136–149. Springer, 2010.

[53] Jonathan Katz. Bridging game theory and cryptography: Recent results and future directions. In *Theory of Cryptography*, pages 251–272. Springer, 2008.

[54] Gillat Kol and Moni Naor. Cryptography and game theory: Designing protocols for exchanging information. In *Theory of Cryptography*, pages 320–339. Springer, 2008.

[55] Yu Kou, Shu Lin, and Marc P. C. Fossorier. Low-density parity-check codes based on finite geometries: a rediscovery and new results. *IEEE Transactions on Information Theory*, 47(7):2711–2736, 2001.

[56] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In *Proceedings of the twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 143–156. SIAM, 2012.

[57] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 2009.

[58] Moses Liskov. Constructing an ideal hash function from weak ideal compression functions. In *Selected Areas in Cryptography*, pages 358–375. Springer, 2007.

[59] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error-correcting Codes*. Elsevier, 1977.

[60] Robert J. McEliece and Dilip V. Sarwate. On sharing secrets and Reed-Solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

[61] Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 650–661. ACM, 2012.

[62] Moni Naor and Guy N Rothblum. The complexity of online memory checking. *Journal of the ACM (JACM)*, 56(1):2, 2009.

[63] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy, 2008. SP 2008.*, pages 111–125. IEEE, 2008.

[64] Martin J Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT press, 1994.

[65] Maura B. Paterson and Douglas R. Stinson. A simple combinatorial treatment of constructions and threshold gaps of ramp schemes. *Cryptography and Communications*, 5(4):229–240, 2013.

[66] Maura B. Paterson, Douglas R. Stinson, and Jalaj Upadhyay. A coding theory foundation for the analysis of general unconditionally secure proof-of-retrievability schemes for cloud storage. *Journal of Mathematical Cryptology*, 7(3):183–216, 2013.

[67] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *Advances in Cryptology–CRYPTO 2010*, pages 502–519. Springer, 2010.

[68] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[69] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In *Advances in Cryptology–EUROCRYPT 2011*, pages 487–506. Springer, 2011.

[70] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 793–802, 2013.

[71] Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM Journal on Discrete Mathematics*, 25(4):1562–1588, 2011.

[72] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *Advances in Cryptology – ASIACRYPT*, pages 90–107, 2008.

[73] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[74] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $O((\log n)^3)$ worst-case cost. In *Advances in Cryptology–ASIACRYPT 2011*, pages 197–214. Springer, 2011.

[75] 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*, pages 325–336. ACM, 2013.

[76] Victor Shoup. *A Computational Introduction to Number Theory and Algebra.* Cambridge university press, 2009.

[77] Daniel A Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings of the twenty-seventh Annual ACM symposium on Theory of Computing*, pages 388–397. ACM, 1995.

[78] Emil Stefanov, Elaine Shi, and Dawn Song. Towards practical oblivious RAM. *arXiv preprint arXiv:1106.3652*, 2011.

[79] D. R. Stinson. *Cryptography: Theory and Practice.* Chapman & Hall/CRC Press Inc., 2006.

[80] Douglas R. Stinson and Jalaj Upadhyay. Is extracting data the same as possessing data? *Journal Mathematical Cryptology*, 8(2):189–207, 2014.

[81] Martin Tompa and Heather Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *28th Annual Symposium on Foundations of Computer Science, 1987*, pages 472–482. IEEE, 1987.

[82] K Ulm. Simple method to calculate the confidence interval of a standardized mortality ratio (smr). *American Journal of Epidemiology*, 131(2):373–375, 1990.

[83] Jalaj Upadhyay. Random projections, graph sparsification, and differential privacy. In *Advances in Cryptology–ASIACRYPT (1)*, pages 276–295, 2013.

[84] Jalaj Upadhyay. Differentially private linear algebra in the streaming model. *arXiv preprint arXiv:1409.5414*, 2014.

[85] Jalaj Upadhyay. Randomness efficient fast-johnson-lindenstrauss transform with applications in differential privacy and compressed sensing. *arXiv preprint arXiv:1410.2470*, 2014.

[86] Marten Van Dijk, Ari Juels, Alina Oprea, Ronald L Rivest, Emil Stefanov, and Nikos Triandopoulos. Hourglass schemes: how to prove that cloud files are encrypted. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 265–280. ACM, 2012.

[87] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *IEEE Proceedings INFOCOM, 2010*, pages 1–9, 2010.

[88] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(5):847–859, 2011.

[89] Sergey Yekhanin. Locally decodable codes. In *Computer Science–Theory and Applications*, pages 289–290. Springer, 2011.

Notation Used in This Thesis

Notation	meaning
Chal	challenge in an audit protocol
d	distance of the encoded message space
d^*	distance of the response code
dist	hamming distance between two vectors
$\mathbb{F}$	a finite field
h	size of a message digest (hash value)
k	length of a message
K	key (in a keyed scheme)
ℓ	size of a challenge
m	message
m_i	message block
$\widehat{m}$	message outputted by the $\mathsf{Extractor}$
M	encoded message
$\mathcal{M}$	message space
$\mathcal{M}^*$	encoded message space
n	length of an encoded message
N	dimension of vectors
$\mathcal{P}$	proving algorithm
q	order of underlying finite field
$\vec{r}_M$	response vector for encoded message M
$\mathbb{R}$	field of real numbers
$\mathcal{R}^*$	response code
Resp	response
s	number of bits of M stored by the adversary
s^*	number of precomputed responses stored by the adversary

s	size of the secret in a ramp scheme
S	secret used in the ramp scheme
$\mathsf{succ}(\mathcal{P})$	success probability of proving algorithm
S	tag (in a keyed scheme)
t	adversary's storage (Chapter Chapter 5)
$\mathbf{x}, \mathbf{y}$	vectors
Γ	challenge space
γ	number of possible challenges
Δ	response space
κ	security parameter
ϵ	threshold probability for proof-of-storage systems
ρ	number of servers in MPoR
ϱ	response function
φ	column sparsity of a matrix
ν	confidence probability of a proof-of-storage system
τ_1, τ_2	parameters of ramp scheme
ζ	row sparsity of a matrix
$\langle \mathbf{x}, \mathbf{y} \rangle$	Inner-product of vectors $\mathbf{x}$ and $\mathbf{y}$

Notation Used in This Thesis

Index