Multi-prover Proof-of-Retrievability

Maura B. Paterson
Department of Economics, Mathematics and Statistics
Birkbeck, University of London, Malet Street, London WC1E 7HX, UK
m.paterson@bbk.ac.uk

Douglas R. Stinson[*]
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada
dstinson@math.uwaterloo.ca

Jalaj Upadhyay
College of Information Science and Technology
Pennsylvania State University, State College, Pennsylvania 16802, U.S.A.
jalaj@psu.edu [†]

March 8, 2016

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) systems.

In this paper, we study multiple server PoR systems. Our contribution in multiple-server PoR systems is as follows.

1. 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). We also motivate the study of confidentiality of the outsourced message.
2. We give 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 show how classical statistical techniques used by Paterson, Stinson and Upadhyay (Journal of Mathematical Cryptology: 7(3)) can be extended to evaluate whether the responses of the provers are accurate enough to permit successful extraction.
3. We also look at one specific instantiation of our construction when instantiated with the unconditionally secure version of the Shacham-Waters scheme (Asiacrypt, 2008). This scheme gives reasonable security and privacy guarantee. We show 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.

[*]D. Stinson's research is supported by NSERC discovery grant 203114-11

[†]Work done while at David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada

1 Introduction

In the recent past, there has been a lot of activity on remote storage and the associated cryptographic problem of integrity of the stored data. This question becomes even more important when there are reasons to believe that the remote servers might act maliciously, i.e., one or more servers can delete (whether accidentally or on purpose) a part of the data since there is a good chance that the data will never be accessed and, hence, the client would never find out! In order to assuage such concerns, one would prefer to have a simple auditing system that convinces the client if and only if the server has the data. Such audit protocols, called *proof-of-retrievability* (PoR) systems, were introduced by Juels and Kaliski [11], and closely related *proof-of-data-possession* (PDP) systems were introduced by Ateniese *et al.* [2].

In a PoR protocol, a client stores a message m on a remote server and keeps only a short private *fingerprint* locally. At some later time, when the client wishes to verify the integrity of its message, it can run an audit protocol in which it acts as a verifier while the server proves that it has the client's data. The formal security of a PoR protocol is expressed in terms of an *extractor* – there exists an extractor with (black-box or non-black box) access to the proving algorithm used by the server to respond to the client's challenge, such that the extractor retrieves the original message given any adversarial server which passes the audits with a threshold probability. Apart from this security requirement, two practical requirements of any PoR system would be to have a reasonable bound on the communication cost of every audit and small storage overhead on both the client and server.

PoR systems were originally defined for the single-server setting. However, in the real world, it is highly likely that a client would store its data on more than one server. This might be 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.

The first such study was initiated by Curtmola *et al.* [9], who 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 client to audit a subset of servers even if some of them collude.

On the other hand, Bowers, Juels, and Oprea [8] 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 servers is involved. For example, none of the works mentioned above addresses 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.

1.1 Our Contributions

In Section 2, we give the formal description of multi-server PoR (MPoR) systems. We state the definitions for worst-case and the average-case secure MPoR systems. We also motivate the privacy requirement and state the privacy definition for MPoR systems. In Section 3, we define various primitives to the level required to understand this paper.

In Section 4, 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 5, 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 by Paterson, Stinson and Upadhyay [16] 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 benefits of an MPoR system is that it provides cross-server redundancy. In the past, this feature has been used by Bowers *et al.* [8] to propose a multi-server system called HAIL. We first note that the constructions in Section 4 and Section 5 do not provide any improvement on the storage overhead of the server or the client. In Section 6, we give a construction based on the Shacham-Waters protocol [17] that allows significant reduction of the storage overhead of the client in the multi-server setting.

1.2 Related Works

The concept of *proof-of-retrievability* is due to Juels and Kaliski [11]. A PoR scheme incorporates a challenge-response protocol in which a verifier can check that a message is being stored correctly, along with an *extractor* that will actually reconstruct the message, given the algorithm of a "prover" who is able to correctly respond to a sufficiently high percentage of challenges.

There are also papers that describe the closely related (but slightly weaker) idea of a *proof-of-data-possession scheme* (PDP scheme), e.g., [2]. A PDP scheme permits the possibility that not all of the message blocks can be reconstructed. Ateniese *et al.* [2] also introduced the idea of using *homomorphic authenticators* to reduce the communication complexity of the system. This scheme was improved in a follow-up work by Ateniese *et al.* [4]. Shacham and Waters [17] later showed that the scheme of Ateniese *et al.* [1] can be transformed into a PoR scheme by constructing an extractor that extracts the file from the responses of the Prover on the audits.

Bowers, Juels, and Oprea [8] extended the idea of Juels and Kaliski [11] 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 [10] 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. Paterson *et al.* [16] studied PoR schemes in the setting of unconditional security, and showed some close connections to error-correcting codes.

There have been some other works that provide proof-of-storage. Ateniese *et al.* [5] used *homomorphic identification schemes* to give efficient proof-of-storage systems. Wang *et al.* [20] gave the first privacy preserving public auditable proof-of-storage systems. We refer the readers to the survey by Kamara and Lauter [12] regarding the architecture of proof-of-storage systems.

Distributed Cloud Computing. 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.* [9]. They studied a multiple replica PDP system, which is the natural generalization of single server PDP system to t servers.

Bowers *et al.* [8] introduced a distributed system that 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.* [3] 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.*

1.3 Comparison with Bowers, Juels, and Oprea

The scheme of Bowers, Juels, and Oprea [8] is closest to our work; however, there are some key differences. We enumerate some of them below:

1. The construction of Bowers, Juels, and Oprea [8] is secure only in the computational setting, while we provide security in the setting of unconditional security.

2. Bowers, Juels, and Oprea [8] 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 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 [8] only consider worst-case scenario).

4. The construction of Bowers, Juels, and Oprea [8] 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 [8] – we require extraction to succeed with probability equal to 1, whereas in [8], extraction succeeds with probability close to 1, depending in part on properties of a certain class of hash functions used in the protocol.

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 single block, we simply call it the file.

2 Security Model of Multi-server PoR systems

The essential components of multi-server-PoR (MPoR) systems are natural generalizations of single-server PoR systems. 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 gives M_i to 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) which 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, Verifier picks an index i, where $1 \leq i \leq \ell$, and engages in a challenge-response protocol with Prover_i. In one execution of challenge-response protocol, Verifier picks a challenge c and gives it to Prover_i, and the prover responds with ϱ. The 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 Verifier accepts the response sent by Prover_i.

7. The 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 the keys, $\{K_i : i \in S\}$), and outputs a message $\widehat{m}$. The 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 protocol. 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 multiple 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 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 single-server 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, for example [16] 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 might be sufficient to guarantee a successful extraction. In other words, it is possible to state (and as we show in this paper, achieve) a security guarantee with respect to the average case success probability over all the proving algorithms.

Definition 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 1.

Example 2.1. *Suppose $\eta = 0.7, \nu = 0$ 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 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 2 are satisfied. Therefore, if the* MPoR *scheme is $(0.7, \nu, \tau)$-average secure, the Extractor will succeed.*

Privacy Guarantee. We mentioned at the start of this section 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 [9]. 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 3. *An* MPoR *system is called t-private if no set $\mathcal{A}$ of adversarial provers of size at most t learns anything about the message stored by the* Verifier*.*

Note that $t = 0$ 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 t 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.

Notation. We fix the letter m for the original message, $\mathcal{M}$ to denote the space from which the message m is picked, and M to denote the encoded message. We fix ν to denote the failure probability of the extractor and η to denote the success probability of a proving algorithm. In this paper, we are mainly interested in the case when $\nu = 0$ for both the worst-case and the average-case security. We use n to denote the number of message blocks, assuming the underlying PoR system breaks the message into blocks.

3 Primitives Used in This Paper

3.1 Ramp Schemes

In our construction, we use a primitive related to secret sharing schemes known as *ramp 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 [6, 18].

It is well-known that the size of each player's 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* [7].

Definition 4. (Ramp Scheme). *Let τ_1, τ_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 3.1. *Suppose the dealer wishes to set up $(2, 4, n)$-ramp scheme with the secret (a_0, a_1). The dealer picks a finite field $\mathbb{F}_q$ with $q > n$ such that $a_0, a_1 \in \mathbb{F}_q$. The dealer picks random elements a_2, a_3 independently from the field $\mathbb{F}_q$ and construct the following polynomial of degree 3 over the finite field $\mathbb{F}_q$: $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$. 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 $q = 17$. 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.*

For completeness, we review some of the basic theory concerning the construction of ramp schemes. Linear codes have been used to construct ramp-schemes for over thirty years since the work of McEliece and Sarwate [14]. We will consider a construction from an arbitrary code in this paper. The following relation between an arbitrary code (linear or non-linear) and a ramp scheme is was shown by Paterson and Stinson [15].

Theorem 3.1. *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 3.2. *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 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 3.2. *One can use a Reed-Solomon code to construct a ramp scheme [14]. Let q be a prime and $1 \leq \mathsf{s} < t \leq n < q$. It is well known that, for a prime q, 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 $\mathbb{F}_q$.*

3.2 Single-prover PoR-system

We start by fixing some notation for PoR schemes that we use throughout the paper. Let Γ be the *challenge space* and Δ be the *response space*. We denote by $\gamma = |\Gamma|$ the size of a challenge space. Let $\mathcal{M}^*$ be the space of all encoded messages. The *response function* $\rho : \mathcal{M}^* \times \Gamma \to \Delta$ computes the response $r = \rho(M, c)$ given the encoded message M and the challenge c.

For an encoded message $M \in \mathcal{M}^*$, we define the *response vector* r^M that contains all the responses to all possible challenges for the encoded message M. Finally, define the *response code* of the scheme to be

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

The codewords in $\mathcal{R}$ are just the response vectors that we defined above. Paterson *et al.* [16] proved the following result for a single-prover PoR scheme.

1. On input $\mathcal{P}$, compute the vector $R' = (r'_c : c \in \Gamma)$, where $r'_c = \mathcal{P}(c)$ for all $c \in \Gamma$ (i.e., for every c, r'_c is the response computed by $\mathcal{P}$ when it is given the challenge c).
2. Find $\widehat{M} \in \mathcal{M}^*$ so that $\mathsf{dist}(R', r^{\widehat{M}})$ is minimised.
3. Output $\widehat{m} = e^{-1}(\widehat{M})$.

Figure 1: Extractor for Theorem 3.3

Theorem 3.3. *Suppose that $\mathcal{P}$ is a proving algorithm for a* PoR *scheme with response code $\mathcal{R}$. If the success probability of the corresponding proving algorithm satisfies* $\mathsf{succ}(\mathcal{P}) \geq 1 - \widetilde{\mathsf{d}}/2\gamma$, *where $\widetilde{\mathsf{d}}$ is the Hamming distance of the code $\mathcal{R}$ and γ is the size of the challenge space, then the extractor described in Figure 1 always outputs the message m.*

If we cast this in the security model defined in Section 1 (Definition 1 and Definition 2), then we have the following theorem.

Theorem 3.4. *Suppose that $\mathcal{P}$ is a proving algorithm for a single server* PoR *scheme with response code $\mathcal{R}$. Then there exists a $(1 - \widetilde{\mathsf{d}}/2\gamma, 0, 1, 1)$-*MPoR *system, where $\widetilde{\mathsf{d}}$ is the Hamming distance of the code $\mathcal{R}$ and γ is the size of the challenge space Γ.*

Paterson *et al.* [16] gave a modified version of the Shacham-Waters scheme which they showed is secure in the unconditional security setting. They argued that, in the setting of unconditionally security, any keyed PoR scheme should be considered to be secure when the success probability of the proving algorithm $\mathcal{P}$, denoted by $\mathsf{succ}(\mathcal{P})$, is defined as the average success probability of the prover over all possible keys (Theorem 3.5). The same reasoning extends to MPoR systems. Therefore, in what follows and in Section 6, 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 computed over all possible keys. We denote the average success probability of a prover $\mathcal{P}$ over all possible keys by $\mathsf{succ}_{\mathsf{avg}}(\mathcal{P})$. Paterson *et al.* [16] showed the following:

Theorem 3.5. *Let $\mathbb{F}_q$ be the underlying field and let $\ell \geq 1$ be the hamming weight of the challenges made by the* Verifier. *Let d be the hamming distance of the space of encoded message, $\mathcal{M}^*$. Suppose that*

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

where $\gamma = q^n$ is the size of the challenge space and d^* is given by

$$\mathsf{d}^* \approx \binom{n}{\ell}(q-1)^\ell - \binom{n-\mathsf{d}}{\ell}(q-1)^\ell - \sum_{w \geq 1} \binom{\mathsf{d}}{w}\binom{n-\mathsf{d}}{\ell - w}\frac{(q-1)^\ell}{q}. \tag{2}$$

Then there exists an Extractor *that always output $\widehat{m} = m$.*

4 Worst-case MPoR Based on 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.

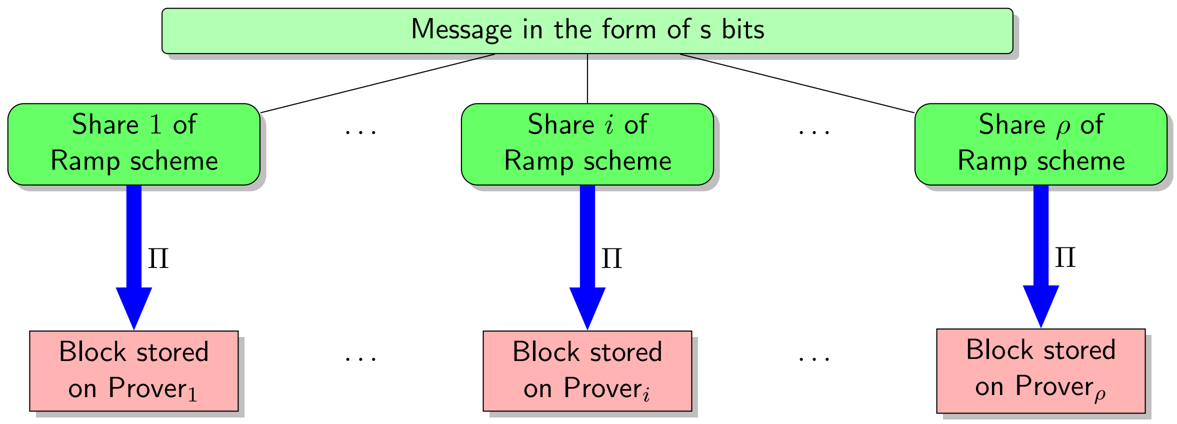
Message in the form of s bits
Share 1 of Ramp scheme
. . .
Share i of Ramp scheme
. . .
Share ρ of Ramp scheme
Π
Π
Π
Block stored on Prover$_1$
. . .
Block stored on Prover$_i$
. . .
Block stored on Prover$_\rho$

Figure 2: Schematic View of Ramp-MPoR System

We first present a schematic diagram of the working of an MPoR in Figure 2 and illustrate the scheme with the help of following example. We provide the details of the construction in Figure 3.

Example 4.1. *Let $\rho = 6$. Suppose the* Verifier *and the provers use a* PoR *system* Π. *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 *picks an encoding function* $e(\cdot)$ *and 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 of dimension* ρ, *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 four provers and sends the challenges to the provers. Once all the provers that he chose reply, he verifies their response. For example, suppose the* Verifier *picks* Prover$_1$, Prover$_3$, Prover$_4$, *and* Prover$_6$. *The* Verifier *then 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 practical deployments of the Ramp-MPoR stated in Figure 3. 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, n)-Ramp scheme 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 3.

Theorem 4.1. *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. *Let* Ramp = (ShareGen, Reconstruct) *be a* (τ_1, τ_2, ρ)*-ramp scheme. Then* Ramp-MPoR, *defined in Figure 3, 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* $\eta = 1 - \widetilde{\mathsf{d}}/2\gamma$.

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

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

Initialization Stage. The $\mathsf{Verifier}$ performs the following steps for storing the message

1. The $\mathsf{Verifier}$ chooses a single-server PoR system Π and a (τ_1, τ_2, ρ)-ramp scheme $\mathsf{Ramp} = (\mathsf{ShareGen}, \mathsf{Reconstruct})$.
2. The $\mathsf{Verifier}$ computes ρ shares of the message using the ramp scheme $(m_1, \ldots, m_\rho) \leftarrow \mathsf{ShareGen}(m)$.
3. The $\mathsf{Verifier}$ 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 $\mathsf{Verifier}$ stores M_i on Prover_i.

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

Figure 3: Worst-case Secure MPoR Using a Ramp-scheme (Ramp-MPoR).

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

1. $\mathsf{Extractor}$ 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. $\mathsf{Extractor}$ 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 3.3) 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 $\mathsf{Extractor}$ will be the message m. $\square$

As a special case of the above, we get a simple MPoR system which uses a *replication code*. A replication code has an encoding function $\mathsf{Enc} : \Lambda \to \Lambda^\rho$ such that $\mathsf{Enc}(x) = (\underbrace{x, x, \ldots, x}_{\rho \text{ times}})$ for any $x \in \Lambda$. This is the setting considered by Curtmola *et al.* [9].

We call a Ramp-MPoR scheme based on a replication code a Rep-MPoR. The schematic description of the scheme is presented in Figure 4 and the scheme is presented in Figure 5. Since a ρ-replication code is a $(0, 1, \rho)$-ramp scheme, a simple corollary to Theorem 4.1 is the following.

Corollary 4.2. *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 Rep-MPoR, formed by instantiating Ramp-MPoR with the replication code based Ramp scheme, is a MPoR system with the following properties:*

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

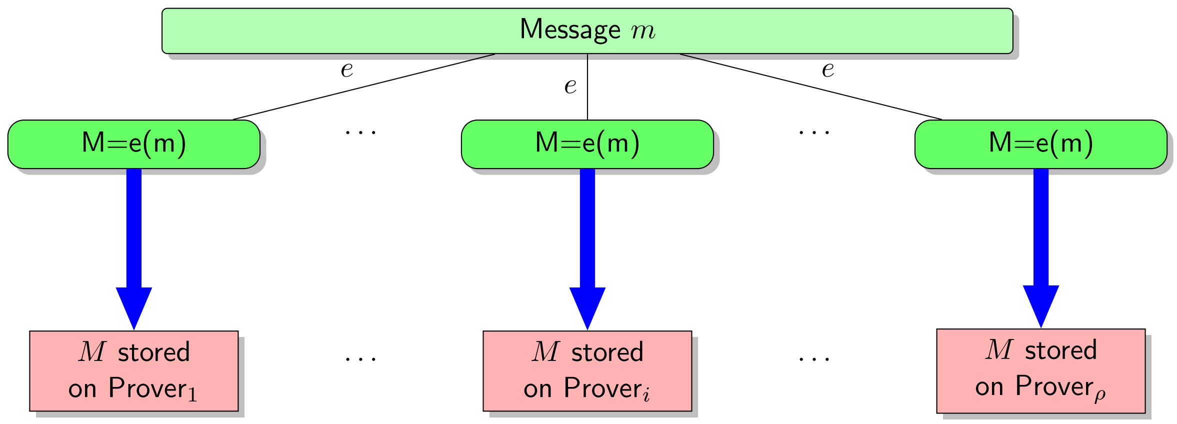

Figure 4: Schematic View of Rep-MPoR

2. *Security: It is $(\eta, 0, 1, \rho)$-threshold secure, where $\eta = 1 - \widetilde{\mathsf{d}}/2\gamma$.*

The issue with Rep-MPoR scheme is that there is no confidentiality of the file. We will come back to this issue later in Section 6.1.

5 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 extraction as described in Section 4. 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 2.1. 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 5.1. *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 5, 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.*

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 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),

Input: The verifier 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.
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 5: Average case Secure MPoR (Rep-MPoR).

2. Compute R as a concatenation of $R_1, \ldots, R_\rho$ and find $\widehat{M} := \left(\widehat{M}_1, \ldots, \widehat{M}_\rho\right)$ 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 $\eta_1, \ldots, \eta_\rho$ 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 5, 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 3.3, it follows that the extraction succeeds if

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

□

5.1 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 Paterson *et al.* [16]. 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 $\mathsf{Pois}(\lambda_1 + \lambda_2 + \ldots + \lambda_\rho)$, even when the λ_i all differ. In the case where the average failure probability is low, the distribution $\mathsf{Pois}(c(f_1 + f_2 + \ldots + f_\rho))$ should provide a reasonable approximation to the actual distribution of the total number of failed challenges.

Example 5.1. *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 1 give both the exact cumulative probability* $\mathsf{Pr}[B \leq b]$ *of observing up to* b *failures, and the Poisson approximation* $\mathsf{Pr}_{\mathsf{Pois}}[B \leq b]$ *of this cumulative probability, for a range of values for* $\mathbf{f}$.

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

Continued on next page ...

Table 1 — continued from previous page

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

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

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

Continued on next page ...

Table 1 — continued from previous page

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	$\mathsf{Pr}[B \le b]$	$\mathsf{Pr}_{\mathsf{Pois}}[B \le b]$
20	0	0.08936904038	0.09536916225
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 $\eta n r$ lies outside this interval. The value of λ_U can be determined easily by exploiting a connection with the chi-squared distribution [19]: 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 Optimization Using the Keyed Shacham-Waters Scheme

In the last three sections, we gave constructions of MPoR scheme 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 [16, 17] for a single server PoR system.

6.1 Extension of the Keyed Shacham-Waters Scheme to MPoR

If we instantiate the Rep-MPoR scheme (described in Section 4) with the modified Shacham-Waters scheme of Paterson *et al.* [16], 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 4.2.

Corollary 6.1. *Let Π be an $(\eta, 0, 0, 1)$-PoR system of Shacham and Waters [17] with a response code of Hamming distance $\widetilde{\mathsf{d}}$ and the size of challenge space γ (where $\widetilde{\mathsf{d}}$ is given by equation (2)). Then Rep-MPoR instantiated with the Shacham-Waters 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, where $\eta = 1 - \frac{\widetilde{\mathsf{d}}(q-1)}{2\gamma q}$.*
3. *Storage Overhead: 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 3.5 with Corollary 4.2. □

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 3, with the Shacham-Waters scheme. We then reduce the storage on the Verifier through two steps.

6.2 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 6.2, 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 6.3, 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 6.1 to say, τ_1-private (as per Definition 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.

We follow with more details. Let $m = (m[1], \ldots, m[k])$ be the message. The Verifier computes the shares of every message block $(m[1], \ldots, m[k])$ 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 $M_i[1], \ldots, M_i[n]$ 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 $a^{(i)}, b_1^{(i)}, \ldots, b_n^{(i)} \in \mathbb{F}_q$ for $1 \leq i \leq \rho$ and computes the tags as follows:

$$S_i[j] = b_j^{(i)} + a^{(i)} M_i[j] \qquad \text{for } 1 \leq i \leq \rho, 1 \leq j \leq n.$$

The verifier gives Prover_i the tuple of encoded messages $(M_i[1], \ldots, M_i[n])$ and the corresponding tags $(S_i[1], \ldots, S_i[n])$. We call this scheme the Basic-MPoR scheme. The following is straightforward from Theorem 4.1.

Theorem 6.2. *Let Π be an $(\eta, 0, 0, 1)$-PoR scheme of Shacham and Waters [17] 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 (2)). Let Ramp be a (τ_1, τ_2, ρ)-ramp scheme. Then Basic-MPoR defined 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. The same issue was encountered by Paterson *et al.* [16], 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 unconditionally secure setting. However, in the case of MPoR, we can improve the storage requirement of the Verifier as shown in the next step.

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.[1] 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

$$b_j^{(i)} = f_j(i) \qquad \text{and} \qquad a_i = g(i)$$

for $1 \leq j \leq n$, and $1 \leq i \leq \rho$. The Verifier then computes the encoded shares and their corresponding tags as in Basic-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 is the formal description of this scheme. For the scheme described in Figure 6, we prove the following result.

Theorem 6.3. *Let* Ramp = (ShareGen, Reconstruct) *be a* (τ_1, τ_2, ρ)*-ramp scheme. Let* Π *be a single-prover Shacham-Waters scheme [17] with a response code of Hamming distance* $\widetilde{\mathsf{d}}$ *and the size of challenge space* γ*. Then* SW-MPoR*, defined in Figure 6, is an* MPoR *system with the following properties:*

1. *Privacy:* SW-MPoR *is* τ_1*-private.*
2. *Security:* SW-MPoR *is* $(\eta, 0, \tau_2, \rho)$*-threshold secure, where* $\eta = 1 - \frac{\widetilde{\mathsf{d}}(q-1)}{2\gamma q}$.
3. *Storage Overhead:* 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 SW-MPoR is straightforward from the secrecy property of the underlying ramp scheme.

For the security guarantee, we have to show an explicit construction of 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 4.1, because of the relation between every message and tag pair. It was noted by Paterson *et al.* [16] 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. Paterson *et al.* [16] noted 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. 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 knowledge to

[1] 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.

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 Ramp = (ShareGen, 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: MPoR Using Optimized Shacham-Waters scheme (SW-MPoR).

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 τ_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 6.4. *Let $t > 0$ be an 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,*

$$\mathsf{Pr}\left[h\left(x_1\right) = y_1 \wedge \ldots \wedge h\left(x_\tau\right) = y_t\right] = \prod_{i=1}^{t} \mathsf{Pr}\left[h(x_{\alpha_i}) = y_i\right]. \tag{3}$$

Since $h(x)$ is uniformly distributed in $\mathbb{F}_q$, the probability computed in equation (3) is actually equal to q^{-t}.

By construction, $g(x)$ is a random polynomial of degree at most $\tau_1 - 1$. Fact 6.4 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 succeed with high enough probability. The description of the Extractor and its analysis is same as that of Theorem 4.1. 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 $\left\{\widehat{M}_{i_1}, \ldots, \widehat{M}_{i_{\tau_2}}\right\}$.
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 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. □

7 Conclusion

In this paper, we studied PoR systems when multiple provers are involved (MPoR). We motivated and defined the security of MPoR in the worst-case (Definition 1) and the average-case (Definition 2) settings, and extended the hypothesis testing techniques used in the single-server setting [16] 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. In the end of this paper, we looked at an optimized version of MPoR system when instantiated with the unconditionally secure version of the Shacham-Waters scheme [17]. 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.

References

[1] 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.

[2] 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.

[3] Giuseppe Ateniese, Özgür Dagdelen, Ivan Damgård, and Daniele Venturi. Entangled cloud storage. *IACR Cryptology ePrint Archive*, 2012:511, 2012.

[4] 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.

[5] 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.

[6] George Robert Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, volume 48, pages 313–317, 1979.

[7] George Robert Blakley and Catherine Meadows. Security of ramp schemes. In *Advances in Cryptology – CRYPTO 1985*, pages 242–268. Springer, 1985.

[8] 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.

[9] 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.

[10] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Theory of Cryptography*, pages 109–127, 2009.

[11] 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.

[12] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In *Financial Cryptography and Data Security*, pages 136–149. Springer, 2010.

[13] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error-correcting Codes*. Elsevier, 1977.

[14] Robert J. McEliece and Dilip V. Sarwate. On sharing secrets and Reed-Solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

[15] 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.

[16] 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.

[17] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *Advances in Cryptology – ASIACRYPT*, pages 90–107, 2008.

[18] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[19] K Ulm. Simple method to calculate the confidence interval of a standardized mortality ratio (smr). *American Journal of Epidemiology*, 131(2):373–375, 1990.

[20] 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.

[21] Sergey Yekhanin. Locally decodable codes. In *Computer Science–Theory and Applications*, pages 289–290. Springer, 2011.

c	challenge
$C^{\perp}$	dual of a code C
d^*	distance of the response code
d	distance of a codeword
$\mathsf{d}^{\perp}$	dual distance of a code
dist	hamming distance between two vectors
$\mathbf{G}$	generator matrix of a code
k	length of a message
K	key (in a keyed scheme)
ℓ	number of message-blocks
m	message
$m[i]$	i-th message block
$\widehat{m}$	message outputted by the $\mathsf{Extractor}$
$\mathcal{M}$	message space
M	encoded message
$M[i]$	i-th encoded message
$M_j[i]$	i-th encoded message on Prover_j
$\mathcal{M}^*$	encoded message space
n	number of provers
N	codeword length
$\mathcal{P}_i$	proving algorithm of i-th Prover
q	order of underlying finite field
r	response
r^M	response vector for encoded message M
S	tag (in a keyed scheme)
$\mathsf{succ}(\mathcal{P})$	success probability of proving algorithm
$\mathcal{R}^*$	response code
Γ	challenge space
γ	number of possible challenges
Δ	response space
φ	column sparsity of a matrix
ρ	response function
τ	privacy threshold
ζ	row sparsity of a matrix

Table 2: Notation used in this paper