Proofs of Retrievability via Hardness Amplification

Yevgeniy Dodis[1], Salil Vadhan[2], and Daniel Wichs[1]

[1] Department of Computer Science, New York University
{dodis,wichs}@cs.nyu.edu
[2] Harvard School of Engineering & Applied Sciences and Center for Research on Computation and Society, Cambridge, MA
salil@eecs.harvard.edu

Abstract. *Proofs of Retrievability (PoR)*, introduced by Juels and Kaliski [JK07], allow the client to store a file F on an untrusted server, and later run an efficient audit protocol in which the server proves that it (still) possesses the client's data. Constructions of PoR schemes attempt to minimize the client and server storage, the communication complexity of an audit, and even the number of file-blocks accessed by the server during the audit. In this work, we identify several different variants of the problem (such as bounded-use vs. unbounded-use, knowledge-soundness vs. information-soundness), and giving nearly optimal PoR schemes for each of these variants. Our constructions either improve (and generalize) the prior PoR constructions, or give the first known PoR schemes with the required properties. In particular, we

- Formally prove the security of an (optimized) variant of the bounded-use scheme of Juels and Kaliski [JK07], without making any simplifying assumptions on the behavior of the adversary.
- Build the first unbounded-use PoR scheme where the communication complexity is linear in the security parameter and which does not rely on Random Oracles, resolving an open question of Shacham and Waters [SW08].
- Build the first bounded-use scheme with *information-theoretic* security.

The main insight of our work comes from a simple connection between PoR schemes and the notion of *hardness amplification*, extensively studied in complexity theory. In particular, our improvements come from first abstracting a purely information-theoretic notion of *PoR codes*, and then building nearly optimal PoR codes using state-of-the-art tools from coding and complexity theory.

1 Introduction

Many organizations and even average computer users generate huge quantities of electronic data. Although advances in hard-disk capacity have mostly kept up, allowing most users to store their data locally, there are many reasons not to do so. Users worried about reliability want to have replicated copies of their files stored remotely in case their local storage fails. Remotely stored data can be made accessible from many locations making it more convenient for many users. Some companies provide useful functionality on remotely stored data using the "software as a service" model. For example, many web-based e-mail services provide tools for searching and managing remotely stored e-mails, making it beneficial for users to store these remotely. Lastly, some organizations

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-00457-5_36

O. Reingold (Ed.): TCC 2009, LNCS 5444, pp. 109–127, 2009.

create large data sets that must be archived for many years, but are rarely accessed and so there is little reason to store such data locally.

PROOFS OF RETRIEVABILITY. One problem with remote storage is that of accountability. If remotely stored data is rarely accessed, how can users be sure that it is being stored honestly? For example, if a remote storage provider experiences hardware failure and loses some data, it might reason that there is no need to notify its clients, since there is a good chance that the data will never be accessed and, hence, the client would never find out! Alternatively, a malicious storage provider might even choose to delete rarely accessed files to save money. To assuage such concerns, we would like a simple auditing procedure for clients to verify that their data is stored correctly.

Such audits, called *Proofs of Retrievability (PoR)*, were first formalized by Juels and Kaliski in [JK07]. In a PoR protocol a client stores a file F on a server an keeps only a very short private verification string locally. Later, the client can run an audit protocol in which it acts as a verifier while the server proves that it possesses the client's data. The security of a PoR protocol is formalized by the existence of an extractor that *retrieves* the original file F from *any* adversarial server that can pass an audit with some reasonable probability. One simple PoR protocol would be for the client to sign the file F and store only the verification key locally. Then, to run an audit, the server would send the file along with the signature. Of course, for practical use, we are interested in schemes with significantly better efficiency. In particular, we want to minimize the communication between the client and the server, and even wish that the amount of data read by the server to run an audit should be much smaller than (essentially independent of) the size of the original file.

In the rest of the introduction, we introduce a general "PoR framework" and show how prior PoR constructions fit into it. We then describe our contributions by showing how to optimize the components of this framework. We also explain a connection between PoR and "hardness amplification", which will allow us to get qualitatively stronger results for some of our schemes.

1.1 The PoR Framework

Our framework consists of two parts: first, we define a purely information-theoretic primitive which we call a *PoR code* and, second, we give several options for converting PoR codes into "full" PoR schemes.

POR CODES. A PoR code consists of three procedures Init, Read and Resp. The function Init specifies the *initial encoding* of the original *client file* F into the *server file* $F' = \mathsf{Init}(F)$ which is stored on the server. The functions $\mathsf{Read}, \mathsf{Resp}$ are used to specify a *challenge-response* audit protocol. The client sends a *random challenge* e which consists of two parts $e = (e_1, e_2)$. The first part of the challenge identifies a set of t indices $(i_1, \ldots, i_t) = \mathsf{Read}(e_1)$, which correspond to t locations in F' that the server should read to compute its response. We refer to t as the *locality parameter* and attempt to minimize it. The server reads the sub-string $x = F'[i_1]||\ldots||F'[i_t]$ of the server file F' and computes a *response* $\mu = \mathsf{Resp}(x, e_2)$, which it sends to the client. The PoR code specifies a natural but incomplete PoR protocol, as depicted in Figure 1.

1. The client starts out with the *client file* F and computes a *server file* $F' = \mathsf{Init}(F)$, to store on the server.
2. To run an audit, the client picks a random challenge $e = (e_1, e_2)$ and sends it to the server.
3. The server reads t locations $(i_1, \ldots, i_t) = \mathsf{Read}(e_1)$ of F', resulting in a sub-string x of length t and sends a response $\mu = \mathsf{Resp}(x, e_2)$ to the client.

Fig. 1. An Incomplete PoR Protocol based on a PoR Code $(\mathsf{Init}, \mathsf{Read}, \mathsf{Resp})$

EXTRACTION PROPERTY. For the security of a PoR code, we want to ensure that any (even *computationally unbounded*) adversary $\mathcal{A}$, which provides the correct value μ with some "reasonable" probability ε, must indeed "know" the file F.Towards that goal we require the existence of a *decoder* $\mathcal{D}$ which decodes the file F given oracle access to some such adversary $\mathcal{A}$. We distinguish between two types of "ε-adversaries": an *ε-erasure* adversary answers correctly with probability at least ε and *does not answer* the rest of the time, while an *ε-error* adversary answers correctly with probability at least ε and can *answer incorrectly* the rest of the time. As the names suggest, there is a clear relation between our problem and the erasure/error decoding of *error-correcting codes (ECC)*. In other words, we can think of the list of all correct responses μ as the challenge e varies as comprising a (possibly exponential) encoding of F and an ε-(erasure/error) adversary as defining a corrupted codeword. The extraction property requires that we construct PoR codes which are (erasure/error)-decodable from an ε fraction of correct responses. Since we want to allow $\varepsilon \ll \frac{1}{2}$, we will need to rely on the notion of *list-decoding* (i.e. the decoder $\mathcal{D}$ outputs a small list of L candidates, one of which is the actual file F) for the case of *errors*. Notice that the functions $\mathsf{Init}, \mathsf{Read}, \mathsf{Resp}$ give our PoR codes a special structure, which is not usually present in general ECCs: for any client file F, the server file $F' = \mathsf{Init}(F)$ allows the server to compute a response μ for a challenge e (i.e. any arbitrary position in the full codeword) efficiently by accessing only t "blocks" of F' for some small t. The server file F' should not be much larger than the original file F, while the full codeword, which consists of all responses μ, could be exponentially long and is never computed or stored in full.

BASIC POR CODE CONSTRUCTION. We now describe a basic PoR code construction, which is the basis of most prior work. The function Init is simply an encoding of F under some appropriate ECC, so that F can be recovered from any δ fraction of the blocks of the server file F'. The challenge is a random t-tuple of locations in F', and the response is the value of F' at those locations. We can think of this in our framework as e_1 explicitly listing t locations, e_2 being empty, and the $\mathsf{Read}, \mathsf{Resp}$ functions being identity. On an intuitive level, in order for the server to "forget" *any* part of F, it must "forget" *at least* $(1-\delta)$-fraction of the blocks of F', in which case it should not be able to respond correctly with probability better than $\varepsilon = \delta^t$ (which can be made exponentially small by choosing a constant $\delta < 1$ and setting t to be proportional to the security parameter). However, this is only intuition and our actual proof needs to work the other way — given an adversarial server that responds correctly with probability $\varepsilon > \delta^t$, we need to decode the original file.

FULL POR SCHEMES. The protocol shown in Figure 1 is incomplete, since the server can give any answer in step 3 and we did not specify how the client decides if to accept or reject the audit! We need to ensure that any adversarial server that passes a single audit with probability ε *is* an ε-(erasure/error) adversary. To that end we can have several possible techniques for converting a PoR code into a full PoR scheme.

1. INFORMATION-THEORETIC BOUNDED-USE POR. The simplest technique is to have the client precompute several random challenge-response pairs $\{(e^{(i)}, \mu^{(i)})\}$ and store them locally before giving F' to the server. Later, in the i-th audit, the client sends the challenge $e^{(i)}$ and directly verifies the server's response by comparing it against the correct stored value $\mu^{(i)}$. To argue the security of this construction, we notice that a prover who can pass an audit with probability ε must be an ε-*error* adversary. The advantage of this solution comes from the fact that it does not need to rely on any computational assumptions, and, hence, we get *information-theoretic security*. The downside comes from the fact that a fresh challenge-response pair is needed for each audit. Thus, we will only get a *bounded-use information-theoretic PoR*, where the client's storage will be proportional to the maximum number of audits ℓ.

2. COMPUTATIONAL BOUNDED-USE POR. It is simple to reduce the client's storage in the above protocol, and make it independent of the number of audits that the client runs, by settling for *computational security*. Firstly, the client picks the challenges $e^{(i)}$ pseudorandomly so that it only needs to remember a short key k_1 for a *pseudorandom function (PRF)* f and can efficiently recreate the challenge $e^{(i)} = f_{k_1}(i)$ later on at the time of an audit. Secondly, the client does not store the responses $\mu^{(i)}$ at all, but instead computes a *tag* $\sigma_i = f_{k_2}((i, \mu^{(i)}))$ for each response, and stores the tags σ_i on the server. During the ith audit protocol, the server computes a a response and sends it along with the ith tag σ_i, so that the client can verify that the response is correct. Note that the client only stores two short keys k_1, k_2 and the rest of the storage is relegated to the server. If the file F is much larger than the maximum number of audits ℓ, the extra server storage will also be relatively insignificant, resulting in a very efficient *bounded-use computational PoR*. The tags σ_i hide future challenge values while ensuring that the server's response is correct, and thus the security analysis is similar to that of the information-theoretic scheme.

3. COMPUTATIONAL UNBOUNDED-USE POR. To get an *unbounded* (computational) PoR scheme, where the client can run an unlimited number of audits, we need a slightly more complicated technique. The basic idea is for the client to provide the server with some *authenticator-tags* for the blocks of F' in addition to the actual server file F', and keep only some small *verification key* locally. The authenticator-tags must be designed specifically to fit the PoR codes in such a way that the server can use them to authenticate the response μ for an *arbitrary* challenge e and convince the client that μ is correct. For example, in the basic PoR code construction, where the response $\mu = F'[i_1]||\ldots||F'[i_t]$ consists of a subset of blocks, the authenticators-tags can simply be the tags of the individual blocks of F' under some *message authentication code (MAC)* so that the server sends the response μ together with the tags of *each of* the blocks $F'[i_1], \ldots, F'[i_t]$. For more complicated PoR codes, we will require smarter authenticator-tag constructions which allow the server to authenticate a short response μ by aggregating the authenticator-tags of the individual blocks in some clever way.

BOUNDED OR UNBOUNDED? Let us compare this last solution with the bounded-use approaches from before. On the positive side, unbounded-use schemes do not force us to choose the number of audits ahead of time. On the negative side, the main problem with the authenticator-based solution is that, in all known schemes, *the authenticators require a significant amount of extra storage on the server*, resulting in a large server storage overhead. Intuitively, each block of the server file F' usually needs to be separately authenticated and therefore the solution usually *doubles* server storage (or increases communication complexity by settling for large blocks). Bounded-use schemes can often be significantly more efficient in terms of server storage, especially when the number of audits to be run is much smaller than the file-size. For example, for a 1 GB file, current unbounded-use schemes only become more efficient than bounded-use ones if the client wants to run more than 33 million audits (at one audit per hour, this won't occur for 3,800 years)![1] Therefore, there is often a good reason to study bounded-use schemes in practice. Moreover, bounded-use schemes also allow us to get *information-theoretic* security (at a cost in client storage).

In summary, there are tradeoffs in parameters and security between bounded and unbounded use schemes, and which one is better depends on the particular application. However, there is also another fundamental difference between these schemes. The use of authenticators in unbounded schemes allows us to restrict our attention to ε*-erasure* adversaries, since the decoder can always detect if the adversary answers incorrectly for any challenge. As we shall see, this will translate to a relatively simple form of (unique-)decoding with a straightforward proof. In bounded-use schemes, the decoder cannot verify if arbitrary responses are correct and therefore we need to analyze the (list-)decoding of PoR codes with respect to an ε*-error* adversary, which will make our analysis more difficult. Although we only require list-decoding for the *PoR code*, we would like the extractor for the *full PoR scheme* to output a single candidate which matches the client's file F with overwhelming probability. To do so, we will also make the client store a short, private, "almost-universal" hash h of the file F and the extractor outputs the only possibility in the list which matches the hash.

1.2 Prior Work

Naor and Rothblum [NR05] studied a primitive called *sublinear authenticators* which is closely related to PoR. Although the motivation for sublinear authenticators is somewhat different than PoR, we can also think of sublinear authenticators as PoR schemes that provide security against an *restricted class of adversarial servers*. In particular, for sublinear authenticators, we assume that the adversarial server runs the *honest code* of an audit protocol, but does so using some possibly *modified* server file $\widetilde{F}' \neq F'$. In the PoR setting, the adversarial server may use an *arbitrary strategy* to run an audit protocol and its responses may not be consistent with *any* particular server file. Hence, security for sublinear-authenticators is strictly weaker and does *not necessarily* imply security for PoR. The main result of Naor and Rothblum is a lower bound for *information theoretic unbounded-use* sublinear authenticators, which translated to a lower-bound for *information theoretic unbounded-use* PoR schemes, essentially showing that

[1] We assume an additional server storage of (at least) 1 GB for the unbounded-use schemes versus 256 bits per audit for bounded-use schemes, on top of the file F'.

schemes of this type *cannot* be efficient. In addition, Naor and Rothblum proposed two constructions for sublinear authenticators: a bounded-use information theoretic scheme (corresponding to construction 1 in our framework) and an unbounded-use computational scheme (corresponding to construction 3). Both schemes use the basic PoR code that we described. In [NR05], these schemes were shown to be secure as sublinear authenticators *but not* as PoR schemes.

Juels and Kaliski [JK07] were the first to define PoR schemes formally and gave two PoR constructions. First, they show that the unbounded-use computational scheme of [NR05] is also secure as a PoR scheme. Second, Juels and Kaliski propose a computational bounded-use scheme as their main construction. This scheme is essentially equivalent to construction 2 within our framework and also uses the basic PoR code. However, to prove the security of the scheme, [JK07] resorts to a simplifying assumption on the behavior of the adversary called *block-independence*, more or less assuming that the PoR attacker follows the restrictive syntax of the sublinear-authenticator attacker mentioned above. Put differently, they only show the security of their scheme as a sublinear authenticator, and make the "simplifying assumption" that this is enough for full PoR security. In a follow-up work, Bowers et al. [BJO08] use a slightly different simplifying assumption, requiring that the adversary cannot update its state during a multi-round protocol. Neither work gives any security guarantees against fully Byzantine adversarial servers.

Shacham and Waters [SW08] notice that, in the *unbounded-use* scheme of [NR05, JK07] (corresponding to construction 3, with basic PoR code), the communication between server and client is unnecessarily large; in particular, it is $O(\lambda^2)$ where λ is the security parameter. This is because a server response consists of $t = O(\lambda)$ file blocks, each of which is of size $O(\lambda)$ (being the tag for the block under the message authentication code). In our language, the problem is that the underlying PoR code has a large *output alphabet*. Shacham and Waters showed that this is not necessary, by constructing a new scheme which implicitly uses an *improved* PoR code with a smaller output alphabet. This scheme improves the server's response size but, unfortunately, at the cost of increasing the client's challenge size to $O(\lambda^2)$ — a problem which was remedied in [SW08] through the use of Random Oracles. The main contribution of Shacham and Waters is the construction of *homomorphic linear authenticators*, following a similar (but informal and less efficient) approach of Ateniese et al. [ABC+07]. Such authenticators allow the server to aggregate the tags of individual file blocks and authenticate a response under the improved PoR code (actually, any linear functions of the blocks) using a single short tag. Shacham and Waters (again following [ABC+07]) also design a PoR scheme with *public verifiability* (i.e. the client need not store any private data), using a clever construction of publicly verifiable homomorphic authenticators.

1.3 Our Results

We make two observations about the prior work. Firstly, although all constructions implicitly use some form of PoR codes, such codes have not been defined or studied explicitly. In particular, [NR05, JK07] use the basic PoR code, while the recent work of [SW08] has a clever but ad-hoc optimization which improves some parameters (the response size) at the cost of others (the challenge size). Secondly, we notice that *none* of

the prior *bounded-use* schemes are known to be to secure against fully Byzantine adversarial server strategies, even though such schemes are often more efficient and practical than unbounded-use constructions in many scenarios.

In this work, we undertake a thorough study of PoR codes and give a construction of PoR codes based on *hitting samplers* and *error-correcting codes*. Since all prior PoR code constructions (which implicitly appeared in prior work) employ sub-optimal variants of these primitives, we can improve the efficiency of all known constructions. In particular, we show how to construct a variant of the computational unbounded-use scheme of [SW08], where the challenge size *and* response size are short, without the need of the random-oracle model. We also show that our optimizations improve the prior bounded-use schemes and allow for a *flexible tradeoff* between the locality t and the server storage overhead, without increasing the communication complexity.

As we have already mentioned, proving the security of bounded-use schemes (information theoretic and computational) relies on the security of PoR codes with respect to ε*-error* adversaries, which was never shown fully in prior work. It turns out that most of the difficulty lies in decoding the basic PoR code. Interestingly, this problem is intimately connected to *hardness amplification* and, more specifically, to *direct product theorems* [Yao82, Lev87, Imp95, GNW95, IW97, Tre03, IJK06, IJKW08]. Informally, direct product theorems state that if a given task T is hard to accomplish (for a certain class of attackers) with probability more than δ, then t independently sampled tasks $T_1, \ldots, T_t$ are hard to *simultaneously* accomplish (for a slightly weaker class of attackers) with probability significantly greater than the "expected" $\varepsilon = \delta^t$. To prove such a statement, one starts with the attacker A who can solve $T_1, \ldots, T_t$ with probability greater than ε, and builds a more complicated attacker B who can solve a single task T with probability more than δ. The connection between hardness amplification and error-decoding for the basic PoR code is as follows: one simply defines T to be the task of predicting a random location of F'. Then, the attacker A above becomes an adversarial server who answers ε-fraction of the t-tuples in F', and the constructed attacker B becomes an "extractor" who recovers a δ-fraction of F' (which should suffice in recovering all of F). Using this connection, we will be able construct an efficient extractor for bounded-use schemes, and, thus, provide the *first formal proofs of security* for such schemes, including the first information-theoretic bounded-use scheme.

In Table 1, we compare the efficiency of our schemes with prior construction. We assume that the client file size is k, the security parameter is λ, and choose to parameterize the schemes by a value γ in the range $1 < \gamma \leq 2$, which allows us to formulate a flexible tradeoff between parameters. In all of the schemes the locality $t = \mathcal{O}(\lambda/(\gamma - 1))$ and, as we will see, all bounded-use schemes achieve a server storage of roughly γk, while the use of authenticators in the unbounded-use schemes roughly doubles the server storage.[2] We see that γ highlights a (necessary) tradeoff between server storage overhead and the locality t. For example, setting $\gamma = 2$ we double the server storage and get locality $t = O(\lambda)$, while setting $\gamma = 1.01$ we can achieve bounded-use schemes were the server only stores 1% of additional information, at the expense of $100\times$ increase in

[2] In prior unbounded-use schemes, one might reduce the overhead by making the block size larger. However, this increases communication (and most other parameters, too). Thus, to keep our comparison fair, we assume fixed block size and do not reflect this tradeoff in Table 1.

locality (but no other parameter degradation). We look at both information-theoretic (I.T.) and computational (Comp) security. We also consider both efficient and inefficient "extractors". Intuitively, an efficient extractor provides "knowledge soundness" (Know), guaranteeing that the adversary stores the file in some reasonable format and can efficiently recover it. An inefficient extractor only provides "information soundness" (Inf), guaranteeing that the server still has all of the "information" in the file, but may store it in some inefficient format. In the table we consider all possibilities with (I.T. / Know) being the strongest security guarantee.

Due to the space constraints, all the proofs are deferred to the full version [DVW09].

Table 1. PoR Schemes: Prior Results and Our Improvements. k is the file size, λ is the security parameter, and $1 < \gamma \leq 2$ is a flexible parameter. All schemes have locality $\mathcal{O}(\lambda/(\gamma-1))$.

Scheme	Bounded?	Security	Server Storage	Client Storage	Challenge	Response
[JK07] †	ℓ-time	Comp/Know	$\gamma k + \mathcal{O}(\ell\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}\left(\frac{\lambda}{\gamma-1}\log k\right)$	$\mathcal{O}\left(\frac{\lambda^2}{\gamma-1}\right)$
[JK07]	No	Comp/Know	$2\gamma k$	$\mathcal{O}(\lambda)$	$\mathcal{O}\left(\frac{\lambda}{\gamma-1}\log k\right)$	$\mathcal{O}\left(\frac{\lambda^2}{\gamma-1}\right)$
[SW08] ‡	No	Comp/Know	$2\gamma k$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$
Our	ℓ-time	I.T./Know	γk	$\mathcal{O}\left(\ell\frac{\lambda}{\gamma-1}\log k\right)$	$\mathcal{O}\left(\frac{\lambda}{\gamma-1}\log k\right)$	$\mathcal{O}(\lambda)$
Our	ℓ-time	I.T./Inf	γk	$\mathcal{O}(\ell\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$
Our	ℓ-time	Comp/Know	$\gamma k + \mathcal{O}(\ell\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}\left(\frac{\lambda}{\gamma-1}\log k\right)$	$\mathcal{O}(\lambda)$
Our	ℓ-time	Comp/Inf	$\gamma k + \mathcal{O}(\ell\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$
Our	No	Comp/Know	$2\gamma k$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$

†= Not proven secure as PoR scheme ‡= Random Oracle Model

2 Preliminaries

We assume the basic familiarity with (linear) error-correcting codes. In particular, the standard notation $[n, k, d]_q$ denotes an error-correcting codes over a q-ary alphabet with minimal (Hamming) distance d, block length n and message k. We also assume familiarity with Reed-Solomon codes, which are $[n, k, n-k+1]_q$-codes for a prime power $q \geq n$. For an integer N, we let $[N]$ denote the set $\{1, \ldots, N\}$. Given a string $F \in \Sigma^N$ we let $F[i] \in \Sigma$ denote the ith symbol of F for $i \in [N]$.

A *hitting sampler*, or just *hitter* for short, provides a randomness-efficient way of sampling t elements. We review the definition and parameters achieved by known efficient constructions (see the survey [Gol97] for more details).

Definition 1. *Let* $\mathsf{Hit} : [M] \to [n]^t$ *be a function and interpret the output* $\mathsf{Hit}(e)$ *as a sample of* t *elements in* $[n]$. *We say that* $\mathsf{Hit}(e)$ hits $W \subseteq [n]$ *if it includes at least one member of* W. *A function* Hit *is a* (δ, ρ)-hitter *if for every subset* $W \subseteq [n]$ *of size* $|W| \geq (1-\delta)n$, $\Pr_{e \leftarrow [M]}[\mathsf{Hit}(e)$ *hits* $W] \geq (1-\rho)$.

A simple hitter construction involves choosing t uniformly random and independent elements of $[n]$. This results in a (δ, ρ)-hitter with *sample complexity* $t = \mathcal{O}(\log(1/\rho)/(1-\delta))$ and *randomness complexity* $\log(M) = t\log(n)$. It is known how to reduce the randomness complexity significantly. Indeed, the survey of Goldreich [Gol97] shows how to achieve the following parameters using a construction based on expander graphs.

Theorem 1. *There exists an efficient hitter family such that, for any integer n and any δ, ρ, we get sample complexity $t = \mathcal{O}(\log(1/\rho)/(1-\delta))$ and randomness complexity $\log(M) = \log(n) + 3\log(1/\rho)$.*

3 PoR Codes

A PoR code consists of a triple of functions $(\mathsf{Init}, \mathsf{Read}, \mathsf{Resp})$ with domains and ranges:

$$\mathsf{Init} : (\Sigma_c)^k \to (\Sigma_c)^n \quad , \quad \mathsf{Read} : [M] \to [n]^t \quad , \quad \mathsf{Resp} : (\Sigma_c)^t \times [n'] \to \Sigma_r$$

for some alphabets Σ_c, Σ_r of sizes q_c, q_r respectively. The function Init is an *initial encoding* which converts a *client file* F into a *server file* $F' = \mathsf{Init}(F)$. We let $\widetilde{k} = \lfloor k \log(q_c) \rfloor$ denote the initial file size in bits. The function $\mathsf{Read}, \mathsf{Resp}$ are used by the server to run an audit. The client picks a challenge $e = (e_1, e_2) \in [N]$ where $N = Mn'$ and we identify $[N]$ with $[M] \times [n']$. The function $\mathsf{Read}(e_1)$ determines a tuple of t positions $(i_1, \ldots, i_t)$ in F' which the server must read to produce the response $\mu = \mathsf{Resp}(F'[(i_1, \ldots, i_t)], e_2)$. The functions $\mathsf{Init}, \mathsf{Read}, \mathsf{Resp}$ are actually used by the client/server and hence we require that they are all polynomial time computable.

For our understanding of PoR codes, it makes sense to think of functions $\mathsf{Read}, \mathsf{Resp}$ as defining a *challenge-response* encoding of the server file F'. Firstly, we can think of the Read function as defining a simpler *direct-product encoding* DPE of the server file F' into the codeword $C' \in ((\Sigma_c)^t)^M$ defined by:

$$C' = \mathsf{DPE}(F') = (F'[\mathsf{Read}(1)], F'[\mathsf{Read}(2)], \ldots, F'[\mathsf{Read}(M)])$$

so that each position of C' consists of a concatenation of t positions in F'. The function $\mathsf{Read}, \mathsf{Resp}$ together define the function $\mathsf{Answer} : (\Sigma_c)^n \times [N] \to \Sigma_r$ as $\mathsf{Answer}(F', e) = \mathsf{Resp}(F'[\mathsf{Read}(e_1)], e_2)$ where $e = (e_1, e_2)$. The *challenge-response encoding* CRE of the server file F' results in a codeword $C \in (\Sigma_r)^N$ defined by:

$$C = \mathsf{CRE}(F') = (\mathsf{Answer}(F', 1), \ldots, \mathsf{Answer}(F', N)) .$$

Note that neither the direct-product encoding C' nor the challenge-response encoding C are ever stored explicitly and hence the functions $\mathsf{DPE}, \mathsf{CRE}$ need not be efficient. In our construction, the values N, M, n' will be exponential. Of course, as is usually the case for error-correcting codes, we want to have a family of PoR codes which allows for many flexible choices of the parameters. In particular, we would like to have a family of codes parameterized by the bit size $\widetilde{k} = k \log(q_c)$ of the initial client file and the security parameter λ.

Definition 2. *For any PoR code, an ε-oracle $\mathcal{O}_F$ is an oracle parameterized by some $F \in (\Sigma_c)^k$ such that, letting $F' = \mathsf{Init}(F)$, $C = \mathsf{CRE}(F')$, we have $\Pr_{e \in_R [N]}[\mathcal{O}_F(e) = C[e]] \geq \varepsilon$. We say that $\mathcal{O}_F$ is an* erasure *oracle if, $\Pr[\mathcal{O}_F(e) \notin \{C[e], \perp\}] = 0$. Otherwise we say that $\mathcal{O}_F$ is an* error *oracle.*

Definition 3. *We say that $(\mathsf{Init}, \mathsf{Read}, \mathsf{Resp})$ is a $(\alpha, \beta, \gamma, t)_{q_c}$-PoR code if the* alphabet size *is q_c,* challenge size *is $\alpha = \log_2(N)$, the* response size *is $\beta = \log_2(|\Sigma_r|)$, the* storage overhead*is $\gamma = n/k$ and the* locality *is t. For a PoR code family, all of these values are functions of the parameters $\widetilde{k}, \lambda$. We say that a PoR code is*

- ε_0-erasure decodable *if there is a oracle-access decoder $\mathcal{D}^{(\cdot)}$ such that, for any ε-erasure oracle $\mathcal{O}_F$, with $\varepsilon \geq \varepsilon_0$, the decoder $\mathcal{D}^{\mathcal{O}_F}(\widetilde{k}, \lambda, \varepsilon)$ outputs F with probability at least $1 - 2^{-\lambda}$.*
- $(\varepsilon_0, L(\cdot))$-error decodable *if there is a oracle-access decoder $\mathcal{D}^{(\cdot)}$ such that, for any ε-error oracle $\mathcal{O}_F$ with $\varepsilon \geq \varepsilon_0$, the decoder $\mathcal{D}^{\mathcal{O}_F}(\widetilde{k}, \lambda, \varepsilon)$ outputs a list of size at most $L(\varepsilon)$ containing the element F, with probability at least $1 - 2^{-\lambda}$.*

For both erasures and errors, we say that the scheme is efficiently decodable *if $\mathcal{D}$ runs in time* $\mathsf{poly}(\widetilde{k}, \lambda, 1/\varepsilon)$.

3.1 Constructions of PoR Codes

In all of our constructions, the initial encoding Init is an $[n, k, d]_{q_c}$ error-correcting code with a "good" distance d over the appropriate alphabet Σ_c. The initial encoding defines the server storage overhead $\gamma = n/k$. For the functions $\mathsf{Read}, \mathsf{Resp}$ we first present a *basic construction* followed by two orthogonal improvements.

BASIC CONSTRUCTION. In the basic construction, the challenge is simply a random t-tuple of positions in F', and the response is the value of F' at those positions. More concretely, the number of challenges is $N = M = n^t$ and the function $\mathsf{Read}(e_1)$ simply identifies the value $e_1 \in [N]$ with the tuples $(i_1, \ldots, i_t) \in [n]^t$. The function Resp does not get any portion e_2 of the challenge and is the identity function on the first argument: $\mathsf{Resp}(x, 1) = x$. Thus, the challenge-response encoding CRE is equivalent here to the direct product encoding DPE. This construction yields an $(\alpha, \beta, \gamma, t)_{q_c}$-PoR code where the challenge size is $\alpha = t \log(n)$, the response size is $\beta = \log(q_r) = t \log(q_c)$. This basic PoR code is implicitly used in the schemes of [NR05, JK07].

IMPROVEMENT 1: FLEXIBLE RESPONSE SIZE. One problem with the basic construction that the response size $\beta = t \log(q_c)$ increases proportionally to the locality t. Indeed, in order to achieve good (list-)decoding, we will see that there is an advantage to having a large alphabet Σ_c: i.e., $\log(q_c) = \Omega(\lambda)$. On the other hand, the locality t must also be (at least) proportional to λ, making $\beta = \Omega(\lambda^2)$. In fact, we already mentioned that there is a necessary tradeoff between the locality t and the server storage overhead γ, making t even larger if one is concerned with minimizing the server storage. Thus, we would like to avoid the dependence of the response size β on t.

We improve our basic construction so that, instead of responding with all of the read blocks $x = F'[i_1]||\ldots||F'[i_t]$, the server responds with a randomly chosen position in an encoding of x under some ECC, which we call a *secondary encoding*. More precisely, we instantiate a secondary encoding Sec which is an $[n', k', d']_{q_r}$ ECC over the alphabet Σ_r of size $|\Sigma_r| = q_r$. We assume that it is easy to compute any one position in the codeword without computing the entire codeword. In particular, we define $\mathsf{Resp} : (\Sigma_r)^{k'} \times [n'] \to \Sigma_r$ so that $\mathsf{Resp}(x, e_2) = \mathsf{Sec}(x)[e_2]$ computes the value in position e_2 of the secondary encoding of x. We require that Resp is efficiently computable (but allow Sec to be inefficient, and n' to be exponential). Also, we assume that $(q_r)^{k'} \geq {q_c}^t$ so that we can easily interpret elements in $(\Sigma_c)^t$ as elements in $(\Sigma_r)^{k'}$. The read function remains unchanged from the basic scheme. This yields a PoR code where $\beta = \log(q_r)$ is flexible and does not need to depend on t. For example, we can set $\Sigma_r = \Sigma_c$ and $k' = t$ so that $(\Sigma_r)^{k'} = (\Sigma_c)^t$ and $\beta = \log(q_c)$. As we will see, such

setting will not degrade the (list-)decoding capabilities too much, as long as we set the value of n' and the alphabet size $\log q_c$ to be (appropriately) exponential in the security parameter λ. With such a setting, even a negligible fraction of the responses in a secondary encoding allow us to (list-)decode the original t-tuple x, more or less bringing us back to the basic construction, while reducing the response size β to be $\mathcal{O}(\lambda)$.

We also note that a variant of this improvement was implicitly used by [SW08], which employed the Hadamard code (over a large alphabet) as the secondary encoding.

IMPROVEMENT 2: REDUCING THE CHALLENGE SIZE. We would also like to get rid of the dependence between t and the challenge size α (currently, $\alpha = t\log(n)$). We do so by using derandomization techniques to choose the indices $(i_1, \ldots, i_t)$. Instead of just choosing these indices uniformly at random, we just use a function Read which is a (δ, ρ)-*hitter* (see Definition 1). Intuitively, hitters are useful in our context since, if an adversarial server "forgets" many blocks of F', then the indices chosen by $\mathsf{Read}(e_1)$ are likely to "hit" at least one such block. We can think of the basic PoR code construction as simply employing a "naive" hitter which is not randomness-efficient.

THE GENERAL CONSTRUCTION AND INSTANTIATIONS. To recap, our construction is parameterized by:

- An initial encoding $\mathsf{Init} : (\Sigma_c)^k \to (\Sigma_c)^n$ which is a $[n, k, d]_{q_c}$-ECC.
- A (δ, ρ)-hitter $\mathsf{Read} : [M] \to [n]^t$.
- A secondary encoding $\mathsf{Sec} : (\Sigma_r)^{k'} \to (\Sigma_r)^{n'}$ which is a $[n', k', d']_{q_r}$-ECC and $q_r^{k'} \geq {q_c}^t$.

Most of our analysis will only use generic properties of the above primitives. However, when discussing parameters, we will rely on the following two concrete instantiations of PoR codes. For simplicity, we hide the concrete constants in the "Big-$\mathcal{O}$" notation. Both instantiations are parameterized by the security parameter λ, the file bit-size $\widetilde{k}$, and the server storage overhead γ, and will ensure locality $t = \mathcal{O}(\lambda/(\gamma - 1))$. In fact, they will use the identical Reed-Solomon Codes for their initial and secondary encodings, setting $q_c = 2^{\mathcal{O}(\lambda)}$, $k = \widetilde{k}/\log(q_c)$ and $n = \gamma k$ for the initial Reed-Solomon code, and $q_r = q_c$, $k' = t$ and $n' = 2^{\mathcal{O}(\lambda)}$ for the secondary Reed-Solomon code (recall, this defines $d = n - k + 1$ and $d' = n' - k' + 1$). In fact, the only difference will be in the choice of the (δ, ρ)-hitter, where $\delta \approx \frac{1}{\gamma}$ is roughly the fraction of the initial encoding sufficient to recover the file and $\rho = 2^{-\Omega(\lambda)}$. The first instantiation will use a randomness-efficient hitter construction from Theorem 1, while the second instantiation will use the "naive" hitter, where the t samples are chosen uniformly at random. As we can see, both hitters will indeed achieve sample complexity $t = \mathcal{O}(\log(1/\rho)/(1-\delta)) = \mathcal{O}(\lambda/(\gamma - 1))$. However, the first "clever" hitter will achieve randomness complexity $\log M = \log n + 3\log(1/\rho) = \mathcal{O}(\lambda)$, while the second "naive" hitter will achieve $\log M = t \log n = \mathcal{O}(\lambda \log(\widetilde{k})/(\gamma - 1))$. We summarize the (easily verified) resulting efficiency parameters below, and then state our main technical theorem.

Parameters: $\lambda, \widetilde{k}$ and γ, where $1 < \gamma \leq 2$ (and $\gamma\widetilde{k} + 1 \leq 2^{\lambda/2}$).
First Instantiation: Our first instantiation is an $(\alpha, \beta, \gamma, t)_{q_c}$-PoR code family with:

$$t = \mathcal{O}\left(\frac{\lambda}{\gamma - 1}\right), \quad \log(q_c) = \mathcal{O}(\lambda), \quad \alpha = \mathcal{O}(\lambda), \quad \beta = \mathcal{O}(\lambda) \tag{1}$$

Second Instantiation: Our second instantiation is a $(\alpha, \beta, \gamma, t)_{q_c}$-PoR code family with:

$$t = \mathcal{O}\left(\frac{\lambda}{\gamma - 1}\right), \quad \log(q_c) = \mathcal{O}(\lambda), \quad \alpha = \mathcal{O}\left(\frac{\lambda \log(\widetilde{k})}{\gamma - 1}\right), \quad \beta = \mathcal{O}(\lambda) \tag{2}$$

Theorem 2. *For appropriately selected constants, our PoR code family instantiations have the following security properties.*

1. *The* first *instantiation is* efficiently $(2^{-\lambda})$-erasure *decodable.*
2. *The* first *instantiation is* inefficiently $(2^{-\lambda}, \mathcal{O}(\lambda/\varepsilon^3))$-error *decodable.*
3. *The* second *instantiation is* efficiently $(2^{-\lambda}, \mathcal{O}(\lambda/\varepsilon^3))$-error *decodable.*

Thus, the first construction achieves (essentially) optimal parameters on all fronts, but in the case of errors is only known to be inefficiently decodable, while the second construction is only marginally suboptimal in the challenge size α, but achieves efficient error decodability instead. The proof of this theorem is given in the full version [DVW09]. In the following subsections, we only briefly sketch the high-level outline for the proof of the two efficient decoding variants. The latter variant for the case of errors will use the state-of-the-art direct product theorem of [IJK06, IJKW08] to remove the "simplifying assumption" on the behavior of the adversary made by [JK07].

3.2 Efficient Erasure Decodability

We now show that our first construction is efficiently erasure decodable. To do so, we assume that both the primary and secondary encodings are efficiently erasure-decodable up to the maximum radii d and d', respectively. This is true of the Reed-Solomon code employed by our concrete instantiation. First, we show how to (efficiently) convert an ε-erasure oracle O_F for the full PoR codeword $C = \mathsf{CRE}(\mathsf{Init}(F))$, into an ε'-erasure oracle $\widetilde{O}_F$ for the direct-product encoding $C' = \mathsf{DPE}(\mathsf{Init}(F))$.

Lemma 1. *Let $c_0 = n' - d' + 1$ and let $\mathcal{O}_F$ be an ε-erasure oracle with $\varepsilon \geq 4(c_0/n')$ for the full PoR codeword $C = \mathsf{CRE}(\mathsf{Init}(F))$. Then there is an (efficient) machine $\mathcal{D}_2^{\mathcal{O}_F}(\lambda, \varepsilon)$ which is an ε'-erasure oracle for the codeword $C' = \mathsf{DPE}(\mathsf{Init}(F))$ where $\varepsilon' \geq \varepsilon/4$. Moreover, on a query $e_1 \in [M]$, the machine $\mathcal{D}_2$ runs in time $\mathsf{poly}(\lambda, 1/\varepsilon)$.*

Now we show how to (efficiently) recover $n-d+1$ values in the server file $F' = \mathsf{Init}(F)$ given access to the ε'-oracle $\widetilde{O}_F$. Using erasure-decoding of the initial code, we can then efficiently recover F.

Lemma 2. *Let $\widetilde{\mathcal{O}}_F$ be an ε'-erasure oracle for the codeword $C' = \mathsf{DPE}(\mathsf{Init}(F)) = (F[\mathsf{Read}(1)], \ldots, F[\mathsf{Read}(M)])$ and assume that the function Read is (δ, ρ)-hitter where $\delta = \frac{d+1}{n}$ and $\varepsilon' \geq 2\rho$. Then there is an (efficient) algorithm $\mathcal{D}_1^{\widetilde{\mathcal{O}}_F}(\widetilde{k}, \lambda, \varepsilon')$ such that $\mathcal{D}_1^{\mathcal{O}_F} = F$ with probability $1 - 2^{-\lambda}$ and $\mathcal{D}_1$ runs in time $\mathsf{poly}(\widetilde{k}, \lambda, 1/\varepsilon')$.*

Putting Lemma 1 and Lemma 2 together we easily get Part 1 of Theorem 2.

3.3 Efficient Error Decodability

We now show that our PoR code is also efficiently error decodable. Unfortunately, this forces us to sacrifice some of our generality. We cannot use a general hitter construction

and must instead rely on the "naive" hitter, which samples the t positions uniformly at random. In addition, we need an efficient list-decodability for the secondary encodings and efficient error-correction for the initial encoding. All these properties are met by our second instantiation. Again, we first show how to convert an ε-error oracle that answers with values in the full encoding $C = \mathsf{CRE}(F')$ (which includes the secondary encoding) into an ε'-error oracle that answers with values in the code $C' = \mathsf{DPE}(F')$.

Lemma 3. *Let $\mathcal{O}_F$ be an ε-error oracle where $\varepsilon \geq 8k'/(n')^{1/4}$ for the full PoR codeword $C = \mathsf{CRE}(\mathsf{Init}(F))$. Then there is an (efficient) machine $\mathcal{D}_2^{\mathcal{O}_F}(\varepsilon, \lambda)$ which accepts queries $e_1 \in [M]$ and is an ε'-error oracle for the codeword $C' = \mathsf{DPE}(F')$, where $\varepsilon' = \varepsilon^3/256$. Moreover, $\mathcal{D}_2$ runs in time $\mathsf{poly}(\widetilde{k}, \lambda, 1/\varepsilon)$.*

Now we need to efficiently list-decode the direct-product code $C' = \mathsf{DPE}(F')$. Furthermore, we need to do so by reading only a small number of position in C', and certainly far fewer than the entire codeword. This is a highly non-trivial problem which is intimately related to *direct product theorems* in hardness amplification. We now identify codewords C', F' with functions $C' : [n]^t \to (\Sigma_c)^t$ and $F' : [n] \to (\Sigma_c)$ which map a position in the codeword to the value of the codeword at that position. Direct product theorems [Yao82, Lev87, Imp95, GNW95, IW97, Tre03, IJK06, IJKW08] essentially show that, if there exists an efficient algorithm which ε-computes the direct product function C' then there also exists an efficient algorithm which δ-computes the original function F'. Unfortunately, most direct product theorems (in particular, [Yao82, Lev87, Imp95, GNW95, IW97, Tre03]) are in the context of circuits and the reductions are *not* fully constructive. Instead, the reductions show the *existence* of some *advice* which, if hardwired into a circuit, would allow it to δ-compute F'. They do not provide a way of efficiently finding such advice (except is special restrictive cases) and, hence, these results are not appropriate for our setting where we need to actually *run the reduction* as an extractor. Fortunately, direct-product theorems for uniform adversaries (where all advice is efficiently self-generated) recently appeared in [IJK06, IJKW08]. Let us restate their main result of Impagliazzo et al. [IJKW08] in our language.

Theorem 3. *(Theorem 1.3 of [IJKW08]) Given an ε'-error oracle $\widetilde{O}_F$ for the direct-product function C', there exists an efficient algorithm $\mathcal{D}_1^{\widetilde{O}_F}$ which outputs a list of L candidate oracle-access functions $g_1^{\widetilde{O}_F}, \ldots, g_L^{\widetilde{O}_F}$ such that, with probability $(1 - 2^{-\lambda})$, there exists an $i \in \{1, \ldots, L\}$ for which the function g_i is a δ-error oracle for the function F'. In particular this means that $\left|\{\, j \mid g_i^{\widetilde{O}_F}(j) = F'[j]\}\right| \geq \delta$. Moreover the functions $g_1, \ldots, g_L$ are efficient, $L = \mathcal{O}\left(\frac{\lambda}{\varepsilon}\right)$ and $\delta = 1 - \mathcal{O}(\log\left(\frac{1}{\varepsilon}\right)/t)$.*

Combining Lemma 3 and Theorem 3, we can then show that our second instantiation of a PoR code family is efficiently list decodable, proving Part 3 of Theorem 2. Note that it is an interesting open problem if we can modify the result of [IJKW08] to work with some hitter having parameters similar to Theorem 1. This would also lead to some nice derandomization results for hardness-amplification and seems like a difficult problem. Indeed, the proof of Theorem 3 relies on certain efficient sampling properties of the naive hitter construction that the construction from Theorem 1 does not have.

4 PoR Schemes from PoR Codes

A PoR scheme consists of a generation algorithm Gen and an audit protocol defined by two ITMs $\mathcal{P}, \mathcal{V}$ for the prover (server) and verifier (client) respectively. All of the algorithms are parameterized by a security parameter λ which we will omit from the description in the sequel. The Gen algorithm is a randomized algorithm which takes as input a file $F \in \{0,1\}^*$ and produces $(\widetilde{F}, \mathsf{ver}) \leftarrow \mathsf{Gen}(F)$. In the audit protocol, the prover $\mathcal{P}$ is given $\widetilde{F}$ and verifier $\mathcal{V}$ is given ver. At the conclusion of the protocol, the verifier outputs a verdict $v \in \{\mathsf{accept}, \mathsf{reject}\}$. In general we allow the verifier to be stateful and update the value of ver during the protocol and thus, for example, keep track of how many proofs have been run. For unbounded-use schemes, we will give constructions where the verifier is stateless. Our definition essentially follows that of [JK07] but is slightly more general.

COMPLETENESS. We require that in any interaction $\left\{\mathcal{P}(\widetilde{F}) \rightleftharpoons \mathcal{V}(\mathsf{ver})\right\}$ between honest prover and and honest verifier, the verifier outputs a verdict $v = \mathsf{accept}$.

SOUNDNESS. We define the soundness game $\mathbf{Sound}^{\mathcal{E}}_{\mathcal{A}}(k, \ell)$ between an adversary $\mathcal{A}$ and a challenger. In the soundness game, the adversary gets to create an adversarial prover and the challenger runs the extractor $\mathcal{E}$ on it.

1. The adversary $\mathcal{A}$ chooses a file $F \in \{0,1\}^k$.
2. The challenger produces $(\widetilde{F}, \mathsf{ver}) \leftarrow \mathsf{Gen}(F)$ and gives $\widetilde{F}$ to $\mathcal{A}$. In addition, the challenger initializes a verifier $\mathcal{V}(\mathsf{ver})$.
3. We first have a *test stage*, where the adversary $\mathcal{A}$ gets protocol access to $\mathcal{V}(\mathsf{ver})$ and can run at most $\ell - 1$ proofs with it. For each proof, the adversary gets the output $v \in \{\mathsf{accept}, \mathsf{reject}\}$ of the verifier $\mathcal{V}$.
4. At the end of the test stage, the adversary produces code for an (probabilistic) ITM prover $\widetilde{\mathcal{P}}$ and gives this code to the challenger.
5. Let $\varepsilon = \Pr\left[\left\{\widetilde{P} \rightleftharpoons \mathcal{V}(\mathsf{ver})\right\} = \mathsf{accept}\right]$ be the success probability of the adversarial prover $\widetilde{P}$, and $\bar{F} = \mathcal{E}^{\widetilde{P}}(\mathsf{ver}, k, \varepsilon)$ be the output of the extractor.

For complexity classes $\mathcal{C}_1, \mathcal{C}_2$, we say that an *unbounded-use* PoR scheme is *sound* if there exists an extractor $\mathcal{E} \in \mathcal{C}_1$ and two negligible functions $\varepsilon_0(\cdot), \varepsilon_1(\cdot)$ such that, for any adversary $\mathcal{A} \in \mathcal{C}_2$ and any polynomials p_1, p_2

$$\Pr\left[\varepsilon > \varepsilon_0(\lambda) \ \wedge \ \bar{F} \neq F \mid \varepsilon, \bar{F} \leftarrow \mathbf{Sound}^{\mathcal{E}}_{\mathcal{A}}(p_1(\lambda), p_2(\lambda))\right] \leq \varepsilon_1(\lambda).$$

We say that an *ℓ-time* PoR scheme sound if the above holds with $p_1(\lambda)$ replaced by ℓ.

The definition guarantees that the adversary cannot "lose" the file and still succeed in an audit. We give four interesting variants of this definition based on the the complexity classes $\mathcal{C}_1, \mathcal{C}_2$ of the extractor and adversary.

- If the definition holds for the class $\mathcal{C}^{\mathsf{all}}$ of all ITM adversaries, we say that the scheme has *information-theoretic security*. Otherwise, if the definition holds for the class $\mathcal{C}^{\mathsf{poly}}$ of all ITMs running in time $\mathsf{poly}(\lambda)$, we say the scheme has *computational security*.
- If the run time of $\mathcal{E}$ is $\mathsf{poly}(1/\varepsilon, k, \lambda)$ then we say that our scheme has *knowledge-soundness*. Otherwise, if there is no bound on the running time of $\mathcal{E}$, we say that the scheme has *information-soundness*.

Remark 1. As in Proofs of Knowledge, the extractor is *not* part of the protocol but rather serves as a thought-experiment that helps us define security. The adversary, after running some arbitrary audits with the verifier, should not be able to cleverly "lose" parts of the file F (represented by construction of the prover $\widetilde{\mathcal{P}}$ on which $\mathcal{E}$ fails) and yet still succeed in the subsequent audit with reasonable probability $\varepsilon \geq \varepsilon_0$. Of course, the adversarial server might correctly run all audits, but still refuse to give the full file back to the client. This attack, unfortunately, cannot be prevented if the audits are significantly shorter than the size of the file. Instead, we are satisfied if the server is guaranteed to *know* the file at the conclusion of an audit; whether or not the server will actually *give* that file back to the client is an orthogonal concern.

Remark 2. The four variants of soundness are all meaningful. For example, information soundness is already a strong notion which ensures that the adversarial server cannot save on space (by deleting a portion of the file F) and still pass an audit. Knowledge soundness is a stronger notion, which also guarantees that the server stores the file F in some efficiently recoverable representation. The strongest notion — information-theoretic security with knowledge soundness — means that any adversary (regardless of computational power) must store the file in some efficiently recoverable representation.

FROM POR CODES TO POR SCHEMES. In the following subsections, we will briefly sketch how to build a secure PoR scheme (for any of the four variants) from an appropriate $(\alpha, \beta, \gamma, t)_{q_c}$-PoR code $(\mathsf{Init}, \mathsf{Read}, \mathsf{Resp})$. Intuitively, the key step of the extractor $\mathcal{E}$ will be to simply run the corresponding decoder $\mathcal{D}$, giving $\mathcal{D}$ oracle access to the adversarial prover $\widetilde{P}$. Then, if $\mathcal{D}$ is efficient, we will get knowledge-soundness; if not, we will only settle for information-soundness. As for the attacker's efficiency, recall that PoR codes are information-theoretically secure. Thus, as long as we do not introduce any additional computationally-secure primitives into the final PoR scheme, the resulting security will be information-theoretic. Also, for *bounded-use* schemes we will be using *error-decodable* PoR codes, while for the unbounded-use schemes we can use *erasure-decodable* schemes. In our presentation below, we will be only concentrating on the main ideas, primarily focusing on justifying the parameters claimed in Section 1.3.

4.1 Bounded-Use Information-Theoretic Schemes

First, we present a very simple and efficient construction of an ℓ-time, information-theoretically secure PoR scheme from an *error*-decodable PoR code. We also use a family of *almost-universal* hash functions $\mathcal{H} = \{h\}$. Recall, a function family $\mathcal{H}$ is ψ-*universal*, if for any inputs $x \neq y$, $\Pr_{h \leftarrow \mathcal{H}}(h(x) = h(y)) \leq \psi$. It is well known that one can construct such families on $\widetilde{k}$-bit messages (say, based on polynomial evaluation at a random point) having the description of h and the output length of h be at most $\log(\widetilde{k}/\psi)$ bits each. We will set the value ψ later. Bellow we give a detailed description, which corresponds to construction 1 from the introduction.

Gen:
- Let $F' = \mathsf{Init}(F)$.
- Choose ℓ uniformly random challenges $e^{(1)}, \ldots, e^{(\ell)}$ with $e^{(i)} \in [N]$ and compute the responses $\mu^{(1)}, \ldots, \mu^{(\ell)}$ where $\mu^{(i)} = \mathsf{Answer}(F', e^{(i)}) = \mathsf{Resp}(F'[\mathsf{Read}(e_1^{(i)})], e_2^{(i)})$.
- Chooses a uniformly random hash function $h \leftarrow \mathcal{H}$ and compute $\omega := h(F)$.
- Set $\widetilde{F} = F'$, counter $i = 1$ and $\mathsf{ver} := \left(\langle (e^{(1)}, \mu^{(1)}), \ldots, (e^{(\ell)}, \mu^{(\ell)}) \rangle, h, \omega, i\right)$

$\mathcal{P}, \mathcal{V}$: To run an audit $i \in \{1, \ldots, \ell\}$, the verifier $\mathcal{V}$ sends $e^{(i)}$ to $\mathcal{P}$. Upon the receipt of a challenge e, the prover $\mathcal{P}$ computes $\mu = \mathsf{Answer}(F', e)$ and sends μ to $\mathcal{V}$. When the verifier $\mathcal{V}$ receives a value μ', it checks if $\mu' = \mu^{(i)}$ and outputs accept if yes and reject otherwise (updating $i := i + 1$ in both cases).

Notice that the function h and the value $\omega = h(F)$ are not even used in the audit! Of course, they are used by the extractor $\mathcal{E}$ instead, to check which of the L possibilities returned by the decoder $\mathcal{D}$ is the actual file F.

Theorem 4. *Let* $(\mathsf{Init}, \mathsf{Read}, \mathsf{Resp})$ *be an* $(\alpha, \beta, \gamma, t)_{q_c}$*-PoR code family which is efficiently (resp. inefficiently)* $(\varepsilon_0\varepsilon_1, L)$-error decodable*, where* ε_1 *is any negligible function. Let* $\mathcal{H}$ *be a* ψ*-universal hash family. Then the above scheme is a information-theoretically secure* ℓ*-time PoR protocol with knowledge (resp. information) soundness error at most* $\max(\varepsilon_0, L\psi + 2^{-\lambda})$*. In addition, the scheme has locality* t*, server storage overhead* γ*, communication complexity* $(\alpha + \beta)$ *and client storage overhead* $\ell(\alpha + \beta) + \mathcal{O}(\log(k \log(q_c)/\psi))$.

PARAMETERS. We can now instantiate this scheme with particular error-decodable PoR codes constructed in Theorem 2 (parts 2 and 3). Notice, for both variants, we have $L = \mathcal{O}(\lambda/\varepsilon^3)$ and can achieve $\varepsilon_0 = 2^{-\lambda}$ (by changing constants, if needed). Thus, we can set $\psi = 2^{-2\lambda}\varepsilon_0^3/\lambda$, so that $\psi L \leq 2^{-2\lambda}$ and the description of h and $h(F)$ are only $\mathcal{O}(\log(k \log(q_c)/\psi)) = \mathcal{O}(\lambda)$ bits long. Then, we get the final PoR soundness $\max(\varepsilon_0, L\psi + 2^{-\lambda}) \approx 2^{-\lambda}$. In fact, comparing the parameters for the efficient and inefficient variant (see Equations (1) and (2)), the only noticeable difference is the client challenge size α, equal to $\mathcal{O}(\lambda)$ for the inefficient case, and $\mathcal{O}(\lambda \log(\widetilde{k})/(\gamma - 1))$ for the efficient case. Overall, we get an ℓ-time information-theoretically secure PoR with knowledge/information soundness, matching the parameters claimed in Section 1.3.

4.2 Reducing Client Storage: Bounded-Use Computational Schemes

Using computational assumptions, we now show that it is possible to "transfer" the client's storage of the ℓ challenge/response pairs to the server. Overall, the client's storage becomes $\mathcal{O}(\lambda)$, and the server storage becomes $O(\ell\lambda)$. Below we give a detailed description of construction 2 as outlined in the introduction. Let $\mathcal{F}_1, \mathcal{F}_2$ be two PRF families, where the output size of $\mathcal{F}_1$ is equal to the client's challenge size α, and the output size size of $\mathcal{F}_2$ is $\mathcal{O}(\lambda)$.

Gen: – Let $F' = \mathsf{Init}(F)$.

- Choose a random function $f_{k_1} \in \mathcal{F}_1$ and compute ℓ challenges $e^{(1)} = f_{k_1}(1), \ldots, e^{(\ell)} = f_{k_1}(\ell)$. Compute the responses $\mu^{(1)}, \ldots, \mu^{(\ell)}$, where $\mu^{(i)} = \mathsf{Answer}(F', e^{(i)})$.
- Choose a random function $f_{k_2} \in \mathcal{F}_2$ and compute $\sigma_1 := f_{k_2}\left(1, \mu^{(1)}\right), \ldots, \sigma_\ell := f_{k_2}\left(\ell, \mu^{\ell}\right)$. Set $\widetilde{F} := (F', \sigma_1, \ldots, \sigma_\ell)$.
- Choose a uniformly random hash function $h \leftarrow \mathcal{H}$ and set $\omega = h(F')$.
- Initialize count $i = 1$ and set $\mathsf{ver} := (k_1, k_2, h, \omega, i)$.

$\mathcal{P}, \mathcal{V}$: To run an audit $i \in \{1, \ldots, \ell\}$, the verifier $\mathcal{V}$ computes $e^{(i)} = f_{k_1}(i)$ and sends $(e^{(i)}, i)$ to $\mathcal{P}$. Upon the receipt of a challenge $e = (e_1, e_2)$ and an index i, the prover $\mathcal{P}$ computes $\mu = \mathsf{Answer}(F', e^{(i)})$ and sends (μ, σ_i) to $\mathcal{V}$. When the verifier $\mathcal{V}$ receives a value μ', σ', it verifier $\sigma' = f_{k_2}(i, \mu')$ and rejects if this check fails. Otherwise, it accepts (updating $i := i + 1$ in either case).

COMMENTS AND PARAMETERS. The analysis of this scheme is very similar to the bounded-use information-theoretic scheme. In particular, consider an adversarial prover $\widetilde{P}$ which answers with probability ε. Then this must be an ε'-*error* adversary where $\varepsilon - \varepsilon'$ is some negligible distinguishing advantage for the PRF families $\mathcal{F}_1, \mathcal{F}_2$. We omit this analysis. It is also easy to see that the computational PoR protocols with information/knowledge soundness, resulting by using our particular error-decodable PoR codes constructed in Theorem 2 (parts 2 and 3), will match the parameters claimed in Section 1.3. In particular, although the client's storage is now reduced to $\mathcal{O}(\lambda)$ even for the case of knowledge soundness, the client's challenge cannot be "pseudorandomly compressed" below $\mathcal{O}(\lambda \log(\widetilde{k})/(1-\gamma))$, since it needs to look random to the server.

4.3 An Unbounded-Use Computational Scheme

We now show how to construct an unbounded use scheme in our framework using the techniques of [SW08]. The construction is based on the concept of a *homomorphic linear authenticator scheme* — a notion we abstract away from the works of [ABC+07, SW08]. On a high level, this is a scheme in which a verifier computes a *vector-tag* $\overline{\sigma} = (\sigma_1, \ldots, \sigma_n)$ for a vector-message $\overline{x} = (x_1, \ldots, x_n)$ consisting of n field values, using some secret key K. A prover, who is given the vector-message $\overline{x}$, the corresponding vector-tag $\overline{\sigma}$, and a vector-challenge $\overline{a} = (a_1, \ldots, a_n)$, but *not the secret key* K, can then efficiently compute an *authenticator* σ for the field element $\mu = \sum_{i=1}^{n} a_i x_i$. The verifier, when given μ', σ' from the prover, can then run a *verification* procedure (using K) and, if it accepts, be convinced that $\mu' = \mu$.

More precisely, a homomorphic authenticator scheme consists of three algorithms $(\mathsf{LinTag}, \mathsf{LinAuth}, \mathsf{LinVer})$, a key domain $\mathcal{K}$ and a field $\mathbb{F}$. For a key $K \in \mathcal{K}$, and a vector-message $\overline{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$ the algorithm $\mathsf{LinTag}_K(\overline{x})$ produces a vector-tag $\overline{\sigma} = (\sigma_1, \ldots, \sigma_n)$. For a vector-challenge of n coefficients $\overline{a} = (a_1, \ldots, a_n)$, the *un-keyed* function $\mathsf{LinAuth}$ computes an authenticator $\sigma = \mathsf{LinAuth}(\overline{x}, \overline{a}, \overline{\sigma})$. Moreover, the $\mathsf{LinAuth}$ algorithm is "local"; i.e., it only reads values x_i, σ_i for which $a_i \neq 0$. Lastly, the verification algorithm computes $b = \mathsf{LinVer}_K(\overline{a}, \mu', \sigma')$, where $b \in \{0, 1\}$, decides if the algorithm *accepts* or *rejects*.

For completeness, we require that for any $K \in \mathcal{K}, \overline{x} \in \mathbb{F}^n, \overline{a} \in \mathbb{F}^n$ letting $\overline{\sigma} \leftarrow \mathsf{LinTag}_K(\overline{x})$, $\sigma \leftarrow \mathsf{LinAuth}(\overline{x}, \overline{a}, \overline{\sigma})$ and $\mu = \sum_{i=1}^{n} x_i a_i$ then $\mathsf{LinVer}_K(\overline{a}, \mu, \sigma) = 1$. For security, given an adversary $\mathcal{A}$, we define the *unforgeability* game as follows:

1. The adversary $\mathcal{A}$ chooses a vector-message $\overline{x} \in \mathbb{F}^n$.
2. The challenger chooses a uniformly random key $K \leftarrow \mathcal{K}$ and computes $\overline{\sigma} \leftarrow \mathsf{LinTag}_K(\overline{x})$. The adversary is given $\overline{\sigma}$.
3. The adversary $\mathcal{A}$ produces a vector-challenge $\overline{a} \in \mathbb{F}^n$ and a tuple (μ', σ').
4. If $\mathsf{LinVer}_K(\overline{a}, \mu', \sigma') = 1$ and $\mu' \neq \sum_{i=1}^{n} a_i x_i$ then the adversary wins.

We require that for every efficient adversary $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the unforgeability game is negligible in the security parameter λ. We refer to [SW08] for a particular, very elegant and efficient construction of such homomorphic linear authenticators. We note that the definition can also be extended to the public-key setting and an efficient construction for such setting was also given in [SW08].

We can employ linear-homomorphic authenticators along with any PoR code in which the response function $\mathsf{Resp} : (\Sigma_c)^t \to (\Sigma_c)$ is *linear* (as is the case in our constructions) to construct a full PoR scheme as follows:

Gen: Let $F' = \mathsf{Init}(F)$. Choose a key K for the linear authenticator scheme and compute $\overline{\sigma} = \mathsf{LinTag}_K(F'[1], \ldots, F'[n])$. Set $\widetilde{F} := (F', \sigma_1, \ldots, \sigma_n)$ and $\mathsf{ver} := K$.

$\mathcal{P}, \mathcal{V}$:
- The verifier $\mathcal{V}$ chooses a uniformly random value $e \in [N]$ and sends e to $\mathcal{P}$.
- The prover $\mathcal{P}$, upon receiving $e = (e_1, e_2)$, computes $(i_1, \ldots, i_t) = \mathsf{Read}(e_1)$ and $\overline{x} = (x_1, \ldots, x_t) = F'[(i_1, \ldots, i_t)] \in (\Sigma_c)^t$. Since the function $\mathsf{Resp}(\overline{x}, e_2)$ linear, we can write $\mu = \mathsf{Resp}(\overline{x}, e_2) = \sum_{i=1}^{t} a_i x_i$ for some coefficients $\overline{a} = (a_1, \ldots, a_t)$. Let $\overline{\sigma} = (\sigma_{i_1}, \ldots, \sigma_{i_t})$.
  The prover computes $\sigma = \mathsf{LinAuth}(\overline{x}, \overline{a}, \overline{\sigma})$ and sends (μ, σ) to $\mathcal{V}$.
- Upon receipt of a value (μ', σ') the client accepts iff $\mathsf{LinVer}_K(\mu', \sigma') = 1$.

In terms of parameters, we see that the client storage is $\mathcal{O}(\lambda)$, client communication is still α, and the server communication is only increased by the the length of the authenticator, which is $\mathcal{O}(\beta)$ (i.e., the size of a field element) for the homomorphic scheme in [SW08]. However, the main price one pays is in the server's storage, to store the tag-vector $\overline{\sigma} = (\sigma_1, \ldots, \sigma_n)$. For the scheme of [SW08], the length of $\overline{\sigma}$ is equal to the length of F, meaning that the server storage is doubled (see also Footnote 2). In terms of security, we get computational *unbounded-use* PoR scheme with *knowledge-soundness*, as long as our PoR code is *efficiently* ε_0*-erasure decodable*, where ε_0 is negligible in λ.

Finally, to obtain our actual parameters claimed in Section 1.3, we use the same linear-authenticator construction as [SW08], but plug in our improved *erasure-decodable* PoR Code from Equation (1) and Theorem 2 (part 1). In particular, by using the (linear) Reed-Solomon code in place of the Hadamard code for the secondary encoding, and a randomness-efficient hitter as our Read function, our construction is an unbounded-use PoR scheme with communication complexity $\mathcal{O}(\lambda)$ in the *standard model*, and without the use of Random Oracles.

Acknowledgments. Yevgeniy Dodis was supported in part by NSF Grants CNS-0831299, CNS-0716690, CCF-0515121, CCF-0133806, and completed part of this work while visiting Center for Research on Computation and Society at Harvard University. Salil Vadhan was supported in part by NSF Grant CNS-0831289.

References

[ABC+07] Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.X.: Provable data possession at untrusted stores. In: Ning, et al. (eds.) [NdVS07], pp. 598–609 (2007)

[BJO08] Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: Theory and implementation. Cryptology ePrint Archive, Report 2008/175 (2008), http://eprint.iacr.org/

[DVW09] Dodis, Y., Vadhan, S., Wichs, D.: Proofs of retrievability via hardness amplification (full version). Cryptology ePrint Archive (2009), http://eprint.iacr.org/

[GNW95] Goldreich, O., Nisan, N., Wigderson, A.: On yao's xor-lemma. Electronic Colloquium on Computational Complexity (ECCC) 2(50) (1995)

[Gol97] Goldreich, O.: A sample of samplers - a computational perspective on sampling (survey). Electronic Colloquium on Computational Complexity (ECCC) 4(20) (1997)

[IJK06] Impagliazzo, R., Jaiswal, R., Kabanets, V.: Approximately list-decoding direct product codes and uniform hardness amplification. In: FOCS, pp. 187–196. IEEE Computer Society, Los Alamitos (2006)

[IJKW08] Impagliazzo, R., Jaiswal, R., Kabanets, V., Wigderson, A.: Uniform direct product theorems: simplified, optimized, and derandomized. In: Ladner, R.E., Dwork, C. (eds.) STOC, pp. 579–588. ACM, New York (2008)

[Imp95] Impagliazzo, R.: Hard-core distributions for somewhat hard problems. In: FOCS, pp. 538–545 (1995)

[IW97] Impagliazzo, R., Wigderson, A.: P = BPP if EXP requires exponential circuits: Derandomizing the xor lemma. In: STOC, pp. 220–229 (1997)

[JK07] Juels, A., Kaliski, B.S.: Pors: proofs of retrievability for large files. In: Ning, et al. (eds.) [NdVS07], pp. 584–597 (2007)

[Lev87] Levin, L.A.: One-way functions and pseudorandom generators. Combinatorica 7(4), 357–363 (1987)

[NdVS07] Ning, P., De Capitani di Vimercati, S., Syverson, P.F.: Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007. ACM, New York (2007)

[NR05] Naor, M., Rothblum, G.N.: The complexity of online memory checking. In: FOCS, pp. 573–584 (2005)

[SW08] Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)

[Tre03] Trevisan, L.: List-decoding using the xor lemma. In: FOCS, pp. 126–135. IEEE Computer Society, Los Alamitos (2003)

[Yao82] Chi-Chih Yao, A.: Theory and applications of trapdoor functions (extended abstract). In: FOCS, pp. 80–91. IEEE, Los Alamitos (1982)