Proofs of Retrievability via Hardness Amplification

Yevgeniy Dodis* Salil Vadhan† Daniel Wichs‡

January 26, 2009

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.

*Computer Science Dept. NYU. Email: dodis@cs.nyu.edu. Supported by NSF grants CNS-0831299, CNS-0716690, CCF-0515121, CCF-0133806. Work partially completed while visiting the Center for Research on Computation and Society at Harvard University.

†School of Engineering and Applied Sciences and Center for Research on Computation and Society, Harvard University. salil@eecs.harvard.edu. Supported by NSF grant CNS-0831289.

‡Computer Science Dept. NYU. Email: wichs@cs.nyu.edu.

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 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.

Figure 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.[1] 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 creates a server file F' which is an encoding of the client file F under some appropriate ECC, so that F can be recovered from any δ fraction of the blocks of the server file F'. Usually δ is some constant (e.g. $\delta = 3/4$) and the size of F' is not much larger than the size of F (e.g. $|F'| \approx 4/3|F|$). The challenge e is a random t-tuple of locations in F', and the response μ 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

[1] We wish to allow for even very small ε, and only assume that it is bounded above some negligible threshold ε_0.

correctly with probability better than $\varepsilon = \delta^t$ (which can be made exponentially small by 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)![2] 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 are only guaranteed list-decoding for the *PoR code*, the extractor for the *full PoR scheme* outputs 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 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).

[2] 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'.

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 more recent 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.

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	$2\gamma k$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$
Scheme 1	ℓ-time	I.T./Know	γk	$\mathcal{O}\left(\ell\frac{\lambda}{\gamma-1}\log k\right)$	$\mathcal{O}\left(\frac{\lambda}{\gamma-1}\log k\right)$	$\mathcal{O}(\lambda)$
Scheme 2	ℓ-time	I.T./Inf	γk	$\mathcal{O}(\ell\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$
Scheme 3	ℓ-time	Comp/Know	$\gamma k + \ell\lambda$	$\mathcal{O}(\lambda)$	$\mathcal{O}\left(\frac{\lambda}{\gamma-1}\log k\right)$	$\mathcal{O}(\lambda)$
Scheme 4 ‡	ℓ-time	Comp/Know	$\gamma k + \ell\lambda$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$
Scheme 5	ℓ-time	Comp/Inf	$\gamma k + \ell\lambda$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$
Scheme 6	No	Comp/Know	$2\gamma k$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$

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

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 $\mathcal{O}(\lambda/(\gamma-1))$.

In Table 1, we compare the efficiency of our schemes (1 – 6) 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 $1 \leq \gamma \leq 2$, 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.[3] We see that γ highlights a (necessary) tradeoff between server storage overhead and the locality t. For example, setting $\gamma = 2$ we double

[3]In prior unbounded-use schemes, one might reduce the overhead by making the block size larger. However, this

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× increase in 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.

A Note on the Setting $\gamma = 1$. We can also set $\gamma = 1$ an thus get no multiplicative server overhead (and only a small additive overhead in most of the above schemes). In this case, there is no locality and the server has to read the entire contents of its storage. However, we do get some additional improvements in our schemes as shown in Table 1 since the client no longer needs to specify to the server which values the server should read (the server always reads all of them)! Therefore, in our Scheme 1, the client storage becomes $O(\ell\lambda)$ and in Schemes 1,3 the challenge size becomes $O(\lambda)$.

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 $q \geq n$. In Appendix A we review few more basic facts about (list-)decodability of codes which will be needed for some proofs. 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 $\rho = (1-\delta)^t$ for any $0 < \delta < 1$. Said differently, for any $0 < \delta < 1, \rho > 0$ we get a (δ, ρ)-hitter with *sample complexity* $t = \mathcal{O}(\log(1/\rho)/(1-\delta))$ and *randomness complexity* $\log(M) = t \log(n)$. Interestingly, the randomness complexity can be reduced significantly by using a more clever construction. Indeed, the survey of Goldreich [Gol97] shows how to achieve the following parameters using a construction based on expander graphs.

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.

Theorem 2. *There exists an efficient hitter family such that, for any integer n and any $0 < \delta < 1, \rho > 0$, 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 $C' \in \left((\Sigma_c)^t\right)^M$ defined by:

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

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:

$$C = \mathsf{CRE}(F') = \left(\mathsf{Answer}(F', 1), \ldots, \mathsf{Answer}(F', N)\right). \tag{2}$$

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 3. *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 4. *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* $1 - 2^{-\lambda}$*.*

- (ε_0, L)-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* $L(\varepsilon, \lambda)$ *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, without making the connection to coding theory explicit.[4]

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. Both instantiations are parameterized by the security parameter λ, the file bit-size $\widetilde{k}$, and the server storage overhead γ, and ensure locality $t = \mathcal{O}(\lambda/(\gamma - 1))$. In fact, both instantiations use the identical Reed-Solomon Codes for their initial and secondary encodings with parameters:

$$\begin{array}{lllll} \text{Initial Encoding:} & q_c = 2^\lambda, & k = \lceil \widetilde{k}/\lambda \rceil, & n = \lceil \gamma k \rceil \\ \text{Secondary Encoding:} & q_r = q_c, & k' = t, & n' = 2^\lambda \end{array} \tag{3}$$

(recall, this defines $d = n - k + 1$ and $d' = n' - k' + 1$). In fact, the only difference between the instantiations is the choice of the (δ, ρ)-hitter Read. The first instantiation uses a randomness-efficient hitter construction from Theorem 2, where we set $\delta = \frac{k}{n} \approx \gamma^{-1}$ and $\rho = 2^{-\lambda}$. Thus we get sample complexity $t = \mathcal{O}(\lambda/(\gamma - 1))$, and randomness complexity $\log M = \log n + 3\log(1/\rho) = \log(n) + 3\lambda$. The second instantiation uses the "naive" hitter, where the $t = \mathcal{O}(\lambda/(\gamma - 1))$ samples are chosen uniformly at random and thus requires a randomness complexity of $\log M = t \log n$. It is easy to see that the second construction is a (δ, ρ)-hitter for any $\delta < 1$ and $\rho = (1 - \delta)^t$.

Below we state the main theorem for our PoR code instantiations. We will show that the the first instantiation (with challenge size close to optimal) is *efficiently* erasure-decodable. We will also show that it is *inefficiently* error-decodable, but we do not know of any efficient error-decoding strategy for this instantiation. Instead, for efficient error-decoding, we are forced to rely on the second instantiation in which the challenge size is somewhat suboptimal.

[4] The concurrent and independent work of [BJO08] makes a similar observation about the use of an inefficient ECC in [SW08] and suggests using different codes. However, they seem to have a very different purpose for this optimization and the final protocols of that work run in many rounds and are not proven secure against fully Byzantine server strategies.

Parametrization: Our two instantiations are parameterized by security parameter λ, file size $\widetilde{k}$ and a flexibility server storage overhead parameter γ, where $1 \leq \gamma \leq 2$. Below we summarize the efficiency of our two PoR code constructions in terms of the challenge size α, the response size β, the alphabet size $\log(q_c)$ and the locality t.

First Instantiation: Our first instantiation is an $(\alpha, \beta, \gamma, t)_{q_c}$-PoR code family with:

$$t = \mathcal{O}\left(\frac{\lambda}{\gamma - 1}\right), \quad \log(q_c) = \lambda, \quad \alpha = \log(n) + 3\lambda, \quad \beta = \lambda$$

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

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

Parameters for $\gamma = 1$: We can get optimal server storage overhead by setting $\gamma = 1$ at the cost of sacrificing locality and having the server read the entire file F'. In this case, there is no point in sending the first part of the challenge $e = (e_1, e_2)$ since the values read by the server are always the same. Therefore, the two instantiation are the same and achieve the following parameters:

$$t = k = \lceil \widetilde{k}/\lambda \rceil, \quad \log(q_c) = \lambda, \quad \alpha = \lambda, \quad \beta = \lambda$$

Figure 2: Two PoR Code Constructions

Theorem 5. *Our PoR code family instantiations have the following security properties.*

1. *The first instantiation is efficiently ε_0-erasure decodable where $\varepsilon_0 = \frac{8\widetilde{k}}{2^\lambda}$.*
2. *The first instantiation is inefficiently (ε_0, L)-error decodable where $\varepsilon_0 = \frac{4\widetilde{k}}{2^{\lambda/2}}$, $L \leq \frac{16\lambda}{\varepsilon^3}$.*
3. *The second instantiation is efficiently (ε_0, L)-error decodable where $\varepsilon_0 = \frac{8\widetilde{k}}{2^{\lambda/4}}$, $L \leq \mathcal{O}\left(\frac{\lambda}{\varepsilon^3}\right)$.*

This theorem will essentially follow from a more general analysis of our PoR framework in the subsequent sections. In particular, the existence of inefficient decoding strategies follows from an analysis of the distance of our PoR code construction as an ECC. In particular, the existence of an inefficient list-decoding strategy under errors (statement 2) follows from the Johnson bound. This analysis appears in Appendix B. Efficient erasure-decodability (statement 1) is analyzed in Section 3.2 and efficient error-decodability (statement 3) is analyzed in Section 3.3.

3.2 Efficient Erasure Decodability

We now show that our general PoR code construction is efficiently erasure decodable, assuming that both the primary and secondary encodings are efficiently erasure-decodable up to their full distances d and d', respectively. In particular, given the correct values of any $n - d + 1$ (respectively $n' - d' + 1$) positions in the codeword, there should be an efficient algorithm which recovers the unique source message whose primary (respectively secondary) encoding yields such values. This is true of the Reed-Solomon code employed by both of our instantiations. 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 codeword $C' = \mathsf{DPE}(\mathsf{Init}(F))$. Recall that the codeword C

consists of a response to each challenge and the codeword C' consists of the blocks of F' accessed for each challenge, as defined in equations (1), (2). The proof of the Lemma appears in Appendix C.1.

Lemma 6. *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. (see Appendix C.2 for proof).

Lemma 7. *Let $\widetilde{\mathcal{O}}_F$ be an ε'-erasure oracle for the codeword $C' = \mathsf{DPE}(\mathsf{Init}(F))$ and assume that the function Read is (δ, ρ)-hitter where $\delta = \frac{n-d+1}{n}$ 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 6 and Lemma 7 together we easily get the following concrete parameters which prove part 1 or Theorem 5.

Lemma 8. *Our general PoR code family is efficiently erasure decodable for $\varepsilon_0 = \max(4(n' - d' + 1)/n', 8\rho)$. Our first concrete instantiation is therefore efficiently erasure decodable for $\varepsilon_0 = \frac{8t}{2^\lambda} \leq \frac{8\widetilde{k}}{2^\lambda}$.*

See Appendix C.3 for proof.

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. Although we can afford some generality, for ease of exposition we just assume that both the primary and secondary codes are appropriately chosen Reed-Solomon codes (as is the case in both of our concrete instantiations). 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 $C' = \mathsf{DPE}(F')$. See Appendix C.4 for proof.

Lemma 9. *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 10. *(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 $\left|\{\ j\ |g_i^{\widetilde{O}_F}(j) = F'[j]\}\right| \geq \delta n$. 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 9 and Theorem 10, we can then show that our second instantiation of a PoR code family is efficiently list decodable, proving Part 3 of Theorem 5. 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 2. This would also lead to some nice derandomization results for hardness-amplification and seems like a difficult problem. Indeed, the proof of Theorem 10 relies on certain efficient sampling properties of the naive hitter construction that the construction from Theorem 2 does not have. Hence, for efficient error decoding, we are forced to use our second instantiation of a PoR code family which uses the naive hitter construction. The following result proves part 3 of Theorem 5 and follows by combining Lemma 9 and Theorem 10. The proof appears in Appendix C.5.

Lemma 11. *Our second instantiation is efficiently (ε_0, L)-error decodable for $\varepsilon_0 = \frac{8\widetilde{k}}{2^{\lambda/4}}$, $L = \mathcal{O}\left(\frac{\lambda}{\varepsilon^3}\right)$.*

4 Defining PoR Schemes

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 $v \in \{0,1\}$ indicating if the verifier accepts ($v = 1$) or rejects ($v = 0$) the audit. 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 roughly 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 $v = 1$.

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. The challenger runs $\left\{\widetilde{\mathcal{P}} \rightleftharpoons \mathcal{V}(\mathsf{ver})\right\}$ and we let $v \in \{0,1\}$ be the verdict output by $\mathcal{V}$. Let $\bar{F} = \mathcal{E}^{\widetilde{P}}(\mathsf{ver}, k, \lambda)$ be the output of the extractor and define w to be 1 if $\bar{F} = F$ and 0 otherwise. Lastly, if $v = 1$, set T to be the running time of the extractor and otherwise set $T = 0$. Define the output of the soundness game to be the tuple (v, w, T).

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

$$\Pr\left[v = 1 \ \wedge \ w = 0 \mid (v, w, T) \leftarrow \mathbf{Sound}^{\mathcal{E}}_{\mathcal{A}}(p_1(\lambda), p_2(\lambda))\right] \leq \nu(\lambda) \tag{4}$$

We say that an *ℓ-time* PoR scheme *sound* if the above holds when the polynomial $p_2(\lambda)$ replaced by the constant 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 $\mathcal{C}_1$ of all ITM adversaries, we say that the scheme has *information-theoretic security*. Otherwise, if the definition holds for the class $\mathcal{C}_1$ of all ITMs running in time $\mathsf{poly}(\lambda)$, we say the scheme has *computational security*.
- We say that the scheme has *knowledge soundness* if the definition holds for the class C_2 of efficient extractors, where an extractor $\mathcal{E}$ is said to be efficient if there is some constant $c > 0$ such that, the expected value $\mathbf{E}[T^c] \leq \mathsf{poly}(k, \lambda)$, where T is a random variable which is defined as the outcome of the exeriment $(v, w, T) \leftarrow \mathbf{Sound}^{\mathcal{E}}_{\mathcal{A}}(p_1(\lambda), p_2(\lambda))$. We also define *information soundness*, which is a weaker notion of soundness that does not restrict the run-time of $\mathcal{E}$.

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 have *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.*

Remark 3. *In our definition of soundness (as in prior definitions of PoR), the extractor also gets access to the client's private storage* ver*. Therefore we should interpret soundness as saying that the server* possesses the necessary data for the client to (efficiently) retrieve the file. *However, we do not necessarily require that the server can uniquely retrieve the client's file on its own (without the client's verification string* ver*). For example, the server may have some small number of L possibilities for the client's file but may be uncertain which of them is the right one. During an audit protocol the server can use the strategy of randomly guessing which of the L options is the client's file and using this choice to compute a response. Therefore the server has a strategy in which it passes an audit with probability $1/L$ but does not uniquely "know" what the client's file is! However, we will make sure that the client's (short) private storage* ver *must resolve any such ambiguity and hence the server must possess enough data for the client to uniquely recover the original file F with overwhelming probability.*

Remark 4. *In our definition of* knowledge *soundness, we measure the efficiency of the extractor by only counting its running time when the prover succeeds in providing a correct proof (and otherwise counting a run-time of 0). This ensures that either the adversary is unlikely to succeed in giving a valid audit, or the expected run-time of the extractor is efficient when the audit succeeds. Our definition of efficiency, requiring that $\mathbf{E}[T^c] = \mathsf{poly}(k, \lambda)$, is based on Levin's definition of average polynomial time (see [Lev84]) which has advantage of being more robust (e.g. to reductions and changes in model of computation) than standard expected polynomial time. Using Markov's inequality, our definition implies that there is a polynomial p such that, for every $\varepsilon > 0$, the probability that $T \geq p(k, \lambda, 1/\varepsilon)$ is at most ε.*

5 Constructions of PoR Schemes from PoR Codes

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.

5.1 Bounded-Use Information-Theoretic Schemes (Schemes 1,2)

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 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 $\log(k/\psi)$ 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. See Appendix D.1 for a proof of the following theorem.

Theorem 12. *Let* $(\mathsf{Init}, \mathsf{Read}, \mathsf{Resp})$ *be an* $(\alpha, \beta, \gamma, t)_{q_c}$*-PoR code family which is* efficiently (resp. inefficiently) (ε_0, L)-error decodable *where* $L(\lambda, \varepsilon) \leq a(\lambda/\varepsilon)^b$ *for some constants* $a, b \geq 1$*. 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 where the precise soundness error is bounded by* $\nu(\lambda) \leq \max(\varepsilon_0, 2a(\lambda 2^\lambda)^b \psi + 2^{-\lambda})$*. In addition, the scheme has locality* t*, server storage* γk*, communication complexity* $(\alpha + \beta)$ *and client storage overhead* $\ell(\alpha + \beta) + \mathcal{O}(\log(k/\psi))$*.*

Parameters for Schemes 1,2 We now show now use this general construction to instantiate our concrete PoR schemes 1,2 (see Table 1). In both schemes we use $\psi = 2^{-c\lambda}$ for some sufficiently large constant c. For scheme 1, which provides information-theoretic security and knowledge soundness (the strongest security notion), we use an efficiently error-decodable PoR code construction (our second instantiation). By part 3 of Theorem 5, this PoR code family is *efficiently* (ε_0, L)-error decodable where $\varepsilon_0 = \frac{8k}{2^{\lambda/4}}$, $L(\lambda, \varepsilon) = O(\frac{\lambda}{\varepsilon^3})$. Therefore, using Theorem 12 we get an information-theoretically secure PoR scheme with knowledge-soundness having a soundness error of $\nu(\lambda) \leq \frac{8k}{2^{\lambda/4}}$. The locality, communication complexity, client storage and server storage of this scheme (see Table 1) all follow from the setting of α, β, q_c in our PoR code (see Figure 2). For scheme 2, which provides information-theoretic security, but only information soundness, we can use the first PoR code construction. Since this construction has an improved challenge size α, we get an improved client storage and challenge size in the full PoR scheme as is shown in Table 1.

5.2 Bounded-Use Computational Schemes (Schemes 3,4,5)

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, the output size of $\mathcal{F}_2$ is $\mathcal{O}(\lambda)$, and the key size for $\mathcal{F}_1, \mathcal{F}_2$ 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).

See Appendix D.2 for a proof of the following theorem.

Theorem 13. *Let* $(\mathsf{Init}, \mathsf{Read}, \mathsf{Resp})$ *be an* $(\alpha, \beta, \gamma, t)_{q_c}$*-PoR code family which is* efficiently (resp. inefficiently) (ε_0, L)-error decodable *where* $\varepsilon_0(\lambda)$ *is negligible and* $L(\lambda, \varepsilon)$ *is polynomial in* $\lambda, 1/\varepsilon$*. Let* $\mathcal{H}$ *be a* ψ*-universal hash family where* $\psi = 2^{-s\lambda}$ *for some sufficiently large constant* s*. Then the above scheme is a* computationally-secure ℓ*-time PoR protocol with* knowledge (resp. information) *soundness. In addition, the scheme has locality* t*, server storage* $\gamma k + \ell\lambda$*, communication complexity* $(\alpha + \beta + \lambda)$ *and client storage overhead* $\mathcal{O}(\lambda) + \mathcal{O}(\log(k/\psi))$.

Parameters for Schemes 3,4,5. For Scheme 3, we instantiate the above construction with the efficient error-decodable PoR code instantiation (the second instantiation) resulting in the parameters given in Table 1. As we see this reduces the client storage compared to the information-theoretic schemes, essentially relegating this storage to the server. Unfortunately, the size of the communicated challenge must still be large since the client must send the full challenge to the server. One way to get around this (while preserving knowledge soundness) is to rely on the random oracle model. The client, instead of sending the challenge in full, can send a random λ-bit nonce which is expanded to the full challenge (by both, client and server) using a hash function modeled as a random oracle. The analysis is preserved since the extractor can just program the random oracle to correspond to actual random challenges.[5] Therefore, in the random oracle model, we can reduce the challenge to a size of λ bits, resulting in Scheme 4 with parameters given in Table 1.[6] Alternatively,

[5] This technique was essentially also used in [SW08] to optimize their unbounded-use computational scheme, though we show that it is not needed for that purpose!

[6] One may also ask if there is some way to use standard computational assumptions to reduce the challenge size. On first thought, it may appear to be sufficient to send a short seed for a PRG and use the PRG to expand this seed into a full challenge. Unfortunately, we see no way to prove such scheme secure since the decoder for a PoR code does not choose its challenges randomly, but needs to be able to sample a random challenge with certain special properties. It does not seem that the decoder could easily sample PRG seeds which, when expanded, result in such challenges.

we can settle for information-soundness and thus use the first instantiation of PoR codes where the challenge size is essentially optimal, but we only have inefficient decodability. This results in Scheme 5, with the parameters shown in Table 1.

5.3 An Unbounded-Use Computational Scheme (Scheme 6)

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 $\mu' = \mu$. This essentially allows the prover in a PoR scheme to show that the a response μ is the correct response to an *arbitrary* challenge e by sending only a short tag σ.

Definition of Homomorphic Authenticators. 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.

In Appendix E, we briefly describe the symmetric key homomorphic linear authenticator from [SW08]. We note that [SW08] also includes a homomorphic linear authenticator scheme in the public key setting which yields a PoR scheme with public verifiability.

PoR Scheme Construction. We can employ linear-homomorphic authenticators along with any PoR code in which the response function $\mathsf{Resp}_{e_2}(x) : (\Sigma_c)^t \to (\Sigma_c)$ defined by $\mathsf{Resp}_{e_2}(x) = \mathsf{Resp}(x, e_2)$, is *linear* (as is the case in our constructions). For simplicity, we just say that the PoR code is *linear* if the above holds. We construct the full PoR scheme out of such PoR codes 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 $\mathsf{LinVer}_K(\mu', \sigma') = 1$.

Theorem 14. *Assume that* $(\mathsf{Init}, \mathsf{Read}, \mathsf{Resp})$ *is an* linear $(\alpha, \beta, \gamma, t)_{q_c}$*-PoR code family which is efficiently* ε_0*-erasure decodable and* $\varepsilon_0(\lambda)$ *is some negligible function. Further assume that* $(\mathsf{LinTag}, \mathsf{LinAuth}, \mathsf{LinVer})$ *is a homomorphic linear authenticator. Then the above construction is a computationally-secure, unbounded-use PoR scheme with knowledge soundness. Moreover the locality is* t*, the server storage is* $\gamma k + (\gamma k/q_c)|\sigma_i|$ *where* $|\sigma_i|$ *is the size of the individual tags output by* LinTag*, the challenge size if* α *and the response size is* $\beta + |\sigma|$ *where* $|\sigma|$ *is the size of the tags output by* $\mathsf{LinAuth}$*.*

The proof essentially follows that of [SW08] (see section 4.1) where it is shown how to extract assuming that the underlying PoR code is erasure-decodable and thus we skip it.

Parameters for Scheme 6 In terms of parameters, 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 3). Finally, to obtain our actual parameters for Scheme 6, as claimed in Table 1 we use the same linear-authenticator construction as [SW08], but plug in our first PoR code instantiation which is efficiently *erasure-decodable*. [7] Therefore, 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, resolving the main open question of [SW08].

References

[ABC+07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Xiaodong Song. Provable data possession at untrusted stores. In Ning et al. [NdVS07], pages 598–609.

[BJO08] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. Cryptology ePrint Archive, Report 2008/175, 2008. http://eprint.iacr.org/.

[GNW95] Oded Goldreich, Noam Nisan, and Avi Wigderson. On yao's xor-lemma. *Electronic Colloquium on Computational Complexity (ECCC)*, 2(50), 1995.

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

[Gur04] Venkatesan Guruswami. *List Decoding of Error-Correcting Codes (Winning Thesis of the 2002 ACM Doctoral Dissertation Competition)*, volume 3282 of *Lecture Notes in Computer Science*. Springer, 2004.

[IJK06] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Approximately list-decoding direct product codes and uniform hardness amplification. In *FOCS*, pages 187–196. IEEE Computer Society, 2006.

[7] In particular, our scheme is essentially equivalent to that of [SW08], but the underlying PoR code is more efficient since it uses the (linear) Reed-Solomon code in place of the Hadamard code for the secondary encoding, and a randomness-efficient hitter as our Read function

[IJKW08] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: simplified, optimized, and derandomized. In Richard E. Ladner and Cynthia Dwork, editors, *STOC*, pages 579–588. ACM, 2008.

[Imp95] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *FOCS*, pages 538–545, 1995.

[IW97] Russell Impagliazzo and Avi Wigderson. $= BPP$ if requires exponential circuits: Derandomizing the xor lemma. In *STOC*, pages 220–229, 1997.

[JK07] Ari Juels and Burton S. Kaliski. Pors: proofs of retrievability for large files. In Ning et al. [NdVS07], pages 584–597.

[Lev84] Leonid A. Levin. Problems, complete in "average" instance. In *STOC*, page 465. ACM, 1984.

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

[NdVS07] Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors. *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*. ACM, 2007.

[NR05] Moni Naor and Guy N. Rothblum. The complexity of online memory checking. In *FOCS*, pages 573–584. IEEE Computer Society, 2005.

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. Cryptology ePrint Archive, Report 2008/073, 2008. http://eprint.iacr.org/.

[Tre03] Luca Trevisan. List decoding using the xor lemma. *Electronic Colloquium on Computational Complexity (ECCC)*, (042), 2003.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91. IEEE, 1982.

A Some Facts from Coding

Let us restate a version of the Johnson bound found in [Gur04].

Theorem 15. *(Johnson Bound) For any $[n, k, d]_q$ code over an alphabet Σ, any word $r \in (\Sigma)^n$, the number of codewords within a distance $e \leq \min(d, n - \sqrt{n(n - d + d/L)})$ of r, is bounded by L.*

We now rephrase this bound in a form that will be easier for us to use.

Theorem 16. *For any $[n, k, d]_q$ code over an alphabet Σ, any $r \in \Sigma^n$ the number of codewords that agree with r in at least an $\varepsilon \geq \sqrt{2\frac{n-d}{n}}$ fraction of positions is bounded by $L = \frac{2}{\varepsilon^2}$.*

Proof. Set $e = (1 - \varepsilon)n$ in Theorem 15. It is easy to see that the theorem is vacuous if $2\frac{n-d}{n} \geq 1$ and otherwise $e \leq (1 - \sqrt{2\frac{n-d}{n}})n \leq (1 - 2\frac{n-d}{n})n \leq 2d - n \leq d$. We are left to show that $e = (1-\varepsilon)n \leq n - \sqrt{n(n - d + d/L)}$ where $L = \frac{2}{\varepsilon^2}$ and our theorem follows from Theorem 15. The above is equivalent to showing $\varepsilon^2 \geq (n - d + d/L)/n$ which follows if $\varepsilon^2 \geq (n - d)/n + 1/L$. It is easy to see that the last requirement holds since $\frac{n-d}{n} \leq \varepsilon^2/2$ and $1/L = \varepsilon^2/2$. □

Theorem 17. *(Reed Solomon List Decoding [Gur04]) Let C be a codeword in a $[n, k, n-k+1]_q$ Reed-Solomon code. Then, given a value $r \in q^n$ such that an $\varepsilon \geq \sqrt{k/n}$ fraction of positions in r agree with C, there is a $\mathsf{poly}(n, \log(q))$ algorithm which produces a list of size $L = \mathcal{O}(n)$ containing C.*

Combining the above with Theorem 16, if $\varepsilon \geq \sqrt{2k/n}$, then we can (efficiently) get a list of size $L \leq \frac{2}{\varepsilon^2}$.

Theorem 18. *(Reed Solomon Unique Decoding) Let C be a codeword in a $[n, k, n-k+1]_q$ Reed-Solomon code. Then, given a value $r \in q^n$ such that an $\varepsilon \geq \frac{1}{2}(1+\frac{k}{n})$ fraction of positions in r agree with C, there is a $\mathsf{poly}(n, \log(q))$ algorithm which uniquely recovers C.*

B The distance of PoR Codes and Inefficient Decoding

In this section, we study our PoR codes as error-correcting codes and analyze their distance. Then, using bounds from coding theory, we show that our PoR codes are (inefficiently) erasure and error decodable.

Lemma 19. *Consider our general PoR code construction based on an initial encoding Init, a hitter Read and a secondary encoding Sec, such that Init is an $[n, k, d]_{q_c}$-ECC, Read $[M] \to [n]^t$ is a (δ, ρ)-hitter for $\delta = \frac{n-d+1}{n}$ and Sec is an $[n', k', d']_{q_r}$-ECC. Then the full challenge response encoding $\mathsf{CRE} \circ \mathsf{Init} : (\Sigma_c)^k \to (\Sigma_r)^N$ has distance $D = (1-\rho)Md'$.*

Proof. Let $F_1, F_2 \in (\Sigma_c)^k$ with $F_1 \neq F_2$. Let $F_1' = \mathsf{Init}(F_1), F_2' = \mathsf{Init}(F_2)$. Then, by the distance of initial encoding, $\Delta(F_1', F_2') \geq d$. Now let $C_1', C_2' \in ((\Sigma_c)^t)^M$ be the codewords

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

for $b \in 1, 2$. Let $A \subseteq [n]$ be the set of positions i in which $F_1'[i] \neq F_2'[i]$ so that $|A| \geq d \geq (1-(n-d+1)/n)n \geq (1-\delta)n$. Then, for any $j \in [M]$ such that $\mathsf{Read}(j)$ hits into A, $C_1'[j] \neq C_2'[j]$. Moreover, by the hitter property of Read, $\Pr_{j \leftarrow_R M}[\mathsf{Read}(j) \text{ hits } A] \geq (1-\rho)$ or, equivalently, there are at least $(1-\rho)M$ values j for which $\mathsf{Read}(j)$ hits A. Therefore, $\Delta(C_1', C_2') \geq (1-\rho)M$. Lastly, let $C_1, C_2 \in (\Sigma_r)^N$ be the challenge-response encodings $C_1 = \mathsf{CRE}(F_1'), C_2 = \mathsf{CRE}(F_2')$. We can think of these as the result of applying the secondary encoding to each position in C_1', C_2' so that each block in C_b' is mapped to a set of n' blocks in C_b. By the distance of the secondary encoding, each position $i \in [M]$ such that $C_1'[i] \neq C_2'[i]$ results in d' positions $j \in [N]$ for which $C_1[j] \neq C_2[j]$. Therefore $\Delta(C_1, C_2) \geq (1-\rho)Md'$ as claimed. □

Lemma 20. *Our general PoR code construction is* inefficiently erasure-decodable *for*

$$\varepsilon_0 = 2\frac{N-D+1}{N} \leq 2\left(\frac{n'-d'+1}{n'} + \rho\right).$$

Proof. The decoder $\mathcal{D}^{\mathcal{O}_F}(\varepsilon, \lambda)$ proceeds in $c = \log_{1-\varepsilon/2}(2^{-\lambda}) = \frac{\lambda}{-\log(1-\varepsilon/2)} \leq \frac{2\lambda}{\varepsilon}$ iterations on each of which it queries $\mathcal{O}_F$ on challenges $e = 1, 2, \ldots, N$. If, in any iteration, it receives more than $N-D+1$ answers, it finds the unique source file F whose encoding agrees with all such answers. Otherwise, it gives up. Assume that $\mathcal{O}_F$ is an ε-erasure oracle with $\varepsilon \geq \varepsilon_0$. The expected number

of answers received in each iteration is then $\varepsilon N \geq \varepsilon_0 N \geq 2(N-D+1)$. By the Markov inequality, the probability that $\mathcal{D}$ receives at least $N-D+1$ answers in a given iteration is then at least $\varepsilon/2$. Lastly, since the iterations are independent from each other, the probability that there exists at least one iteration in which $\mathcal{D}$ receives that many answers is then at least $1-(1-\varepsilon/2)^c \geq 1-2^{-\lambda}$. Hence, if $\mathcal{D}$ succeeds with probability at least $1-2^{-\lambda}$ for $\varepsilon_0 \geq \frac{N-D+1}{N}$ as we wanted to show. For the last part of our claim, we use the distance calculation from Lemma 19 to plug in for D. □

Corollary 1. *Our first instantiation is* inefficiently *erasure decodable for* $\varepsilon_0 = \frac{2(t+1)}{2^\lambda} \leq \frac{2(\widetilde{k}+1)}{2^\lambda}$.

Proof. This follows since the first construction achieves $\rho = 2^{-\lambda}$ and $\frac{n'-d'+1}{n'} \leq \frac{k'}{n'} \leq \frac{t}{2^\lambda}$. □

Lemma 21. *Our general PoR code construction is* inefficiently (ε_0, L)-error decodable *for* $\varepsilon_0 = 2\sqrt{2\frac{N-D}{N}} \leq 2\sqrt{2\left(\frac{n'-d'+1}{n'}+\rho\right)}$ *and* $L \leq \frac{16\lambda}{\varepsilon^3}$.

Proof. As before, the decoder $\mathcal{D}^{\mathcal{O}_F}(\varepsilon, \lambda)$ proceeds in $c = \log_{1-\varepsilon/2}(2^{-\lambda}) \leq \frac{2\lambda}{\varepsilon}$ iterations on each of which it queries $\mathcal{O}_F$ on all challenges $e = 1, 2, \ldots, N$ to construct some *candidate codeword* r. It then finds a list of size at most $L' \leq \frac{2}{(\varepsilon/2)^2} \leq \frac{8}{\varepsilon^2}$ of source messages whose encodings agree with r in at least $\varepsilon/2 \geq \sqrt{2\frac{N-D}{N}}$ positions (see Theorem 16). The algorithm $\mathcal{D}$ merges the lists from all of the iterations together to output a list of size at most $L = cL' \leq 16\lambda/\varepsilon^3$.

Now we need to analyze the probability that F appears in the final list. Firstly, (as in the erasure case) the probability that, on any specific iteration, $\mathcal{O}_F$ answers more than $\varepsilon/2$ fraction of the challenges correctly is at least $\varepsilon/2$. If this event occurs, then F will be in the final list. As for the erasure case, the probability that this occurs in at least one iteration is at least $1-(1-\varepsilon/2)^c \geq 1-2^{-\lambda}$. Hence the decoder $\mathcal{D}$ succeeds with probability $1-2^{-\lambda}$ for any ε-error oracle $\mathcal{O}_F$ with $\varepsilon \geq \varepsilon_0$. The last inequality is derived as in the proof of Lemma 20. □

By plugging in the bound $\frac{n'-d'+1}{n'} + \rho \leq \frac{t+1}{n'}$ four our first PoR code instantiation, we get the following corollary.

Corollary 2. *Our first instantiation is* inefficiently (ε_0, L)-error decodable *where* $\varepsilon_0 = \frac{2\sqrt{2(t+1)}}{2^{\lambda/2}} \leq \frac{3(\widetilde{k}+1)}{2^{\lambda/2}}$ *and* $L \leq \frac{16\lambda}{\varepsilon^3}$.

C Proofs for Efficient Erasure and Error Decoding

C.1 Proof of Lemma 6

Proof. Set $c = \max((4/\varepsilon)c_0, (32/\varepsilon))$. On a query $e_1 \in [M]$, the machine $\mathcal{D}_2^{\mathcal{O}_F}$ works as follows:

1. Set $Q := \emptyset$.
2. For $i = 1, \ldots, c$:
   Choose $e_2^{(i)} \leftarrow_R [n']$. Let $e = (e_1, e_2^{(i)})$ and $\mu = \mathcal{O}_F(e)$.
   If $\mu \neq \perp$, and $|Q| < c_0$, set $Q := Q \cup \{(e_2^{(i)}, \mu)\}$.
3. If $|Q| \geq c_0$ then decode $F[\mathsf{Read}(e_1)]$. Otherwise return $\perp$.

By the correctness of the decoding algorithm for the secondary code and the fact that $\mathcal{O}_F$ is an erasure oracle, we can easily see that $\Pr\left[\mathcal{D}_2^{\mathcal{O}_F}(e_1) \notin \{C'[e_1], \perp\}\right] = 0$. Let us now analyze $\Pr_{e_1 \in [M]}\left[\mathcal{D}_2^{\mathcal{O}_F}(e_1) = C'[e_1]\right]$. We call a value $\mathbf{e_1} \in [M]$ *good* if

$$\Pr_{e_2 \in [n']}[\mathcal{O}_F(\mathbf{e_1}, e_2) = C[(\mathbf{e_1}, e_2)]] \geq \varepsilon/2$$

and *bad* if the above does not hold. Let us define several events and random variables to aid us with the analysis:

- Let E_G be the event that the value e_1 is good.
- Let $E_A^{(i)}$ be an indicator R.V. which is 1 if $\mu \neq \perp$ in iteration i and 0 otherwise.
- Let $E_B^{(i)}$ be the event that a collision occurs in iteration i: i.e. some value $(e_2^{(i)}, \cdot)$ is already in Q.
- Let X_i be an indicator variable which is 1 if the size of the set Q increases or $|Q| \geq c_0$ in iteration i.
- Let E_S be the event that decoding is successful or, alternatively, that in step 3, $|Q| \geq c_0$.

Then

$$\varepsilon = \Pr[\mathcal{O}_F(e) = C[e] \mid E_G]\Pr[E_G] + \Pr[\mathcal{O}_F[e] = C[e] \mid \neg E_G](1 - \Pr[E_G]) \leq \Pr[E_G] + \varepsilon/2$$

and so $\Pr[E_G] \geq \varepsilon/2$. Now, we will analyze several probabilities conditioned on the event E_G. For ease of exposition, we use the notation $\Pr_G[\cdot]$ to denote the conditional probability $\Pr[\cdot \mid E_G]$. Firstly, the events $E_A^{(1)}, \ldots, E_A^{(c)}$ are independent and $\Pr_G[E_A^{(i)}] \geq \varepsilon/2$ for each i. Also, for each $i \in [c]$,

$$\Pr_G[X_i = 1] \geq \Pr_G[E_A^{(i)} \wedge \neg E_B^{(i)}] \geq \Pr[E_A^{(i)}] - \Pr[E_B^{(i)}] \geq \varepsilon/2 - \frac{c_0 - 1}{n'} \geq \varepsilon/4.$$

Let $p = \varepsilon/4$. Although the events X_i are not independent (the probability of a collision increases as Q grows), the probability of $X_i = 1$ is at least p conditioned on any values of $X_1, \ldots, X_{i-1}$. Let us define a family of independent random variable $Z_1, \ldots, Z_c$ taking values in $\{0, 1\}$ such that $\Pr[Z_i = 1] = p$. Then, it is easy to see, that for any value v, $\Pr_G[\sum X_i \leq v] \leq \Pr[\sum Z_i \leq v]$. Now, using the Chernoff-Hoeffding bound, we get

$$\begin{aligned}
\Pr_G\left[\sum_{i=1}^{c} X_i < c_0\right] &\leq \Pr\left[\sum_{i=1}^{c} Z_i < c_0\right] \\
&\leq \Pr\left[\sum_{i=1}^{c} Z_i \leq (1 - 1/2)(pc)\right] \leq e^{-\frac{1}{8}(pc)} \\
&\leq e^{-\frac{\varepsilon}{32}c} \leq e^{-1} \leq 1/2
\end{aligned}$$

Lastly, $\Pr_G[E_S] = \Pr_G[\sum X_i \geq c_0] \geq 1 - 1/2 \geq 1/2$ and $\Pr[E_S] \geq \Pr[E_S \mid E_G]\Pr[E_G] \geq \varepsilon/4$. Hence $\mathcal{D}_2$ is an $\varepsilon' = \varepsilon/4$-erasure oracle for the codeword C'. $\square$

C.2 Proof of Lemma 7

Proof. We first use $\widetilde{\mathcal{O}}_F$ to recover enough positions in the codeword F' and then use an efficient erasure-decoding algorithm for the primary encoding to recover F. Let $c_1 = n - d + 1$ and $c = \max(4c_1/\varepsilon', 16\lambda/\varepsilon')$. The algorithm $\mathcal{D}_1$ works as follows:

1. Let $\widetilde{F'} = (\perp, \ldots, \perp)$.
2. For $i = 1, \ldots, c$:
   Choose $e_1 \leftarrow_R [M]$ and let $\mu = \widetilde{\mathcal{O}}_F(e_1)$. If $\mu = \perp$, then go to the next iteration.
   Else $\mu \in (\Sigma_c)^t$. Let $\mathsf{Read}(e_1) = (j_1, \ldots, j_t) \in [n]^t$. Set $\widetilde{F'}[j_1] := \mu[1], \ldots, \widetilde{F'}[j_t] := \mu[t]$.
3. If there are at least c_1 positions $i \in [n]$ such that $\widetilde{F'}[i] \neq \perp$, then use the erasure-decoding algorithm for the initial encoding to recover F from $\widetilde{F}'$.

Let us define several events and random variables to aid us with the analysis:

- Let $E_A^{(i)}$ be an indicator R.V. which is 1 if $\mu \neq \perp$ in iteration i and 0 otherwise.
- Let $E_B^{(i)}$ be the event that $\widetilde{F}'$ is defined (i.e. not $\perp$) for all positions $\mathsf{Read}(e_1)$.
- Let X_i be an indicator variable which is 1 if the number of positions in which $\widetilde{F}'$ is defined increases in iteration i, or is already at least c_1.
- Let E_S be the event that decoding is successful or, alternatively, that in step 3, $\widetilde{F}'$ is defined in at least c_1 positions.

First, since Read is a $(\delta = \frac{n-d+1}{n}, \rho \leq \varepsilon'/2)$-hitter, the probability that in iteration i, $\mathsf{Read}(e_1)$ falls entirely into the set of at most $c_1 \leq \delta n$ positions of $\widetilde{F'}$ that are defined is at most ρ. Therefore, $\Pr[E_B^{(i)}] \leq \rho \leq \varepsilon'/2$ even conditioned on any possible run of the first $i-1$ iterations of the algorithm.

The events $E_A^{(1)}, \ldots, E_A^{(c)}$ are independent and $\Pr[E_A^{(i)}] \geq \varepsilon'$ for each i. Also, for each $i \in [c]$,

$$\Pr[X_i = 1] \geq \Pr[E_A^{(i)} \wedge \neg E_B^{(i)}] \geq \Pr[E_A^{(i)}] - \Pr[E_B^{(i)}] \geq \varepsilon' - \rho \geq \varepsilon'/2.$$

Let $p = \varepsilon'/2$. Although the events X_i are not independent (the probability that of $E_B^{(i)}$ increases as positions are learned), the probability of $X_i = 1$ is at least p conditioned on any values of $X_1, \ldots, X_{i-1}$. Let us define a family of independent random variable $Z_1, \ldots, Z_c$ taking values in $\{0, 1\}$ such that $\Pr[Z_i = 1] = p$. Then, it is easy to see, that for any value v, $\Pr[\sum X_i \leq v] \leq \Pr[\sum Z_i \leq v]$. Now, using the Chernoff-Hoeffding bound, we get

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{c} X_i < c_1\right] &\leq \Pr\left[\sum_{i=1}^{c} Z_i < c_1\right] \\
&\leq \Pr\left[\sum_{i=1}^{c} Z_i \leq (1 - 1/2)(pc)\right] \leq e^{-\frac{1}{8}(pc)} \\
&\leq e^{-\frac{\varepsilon}{16}c} \leq e^{\lambda} \leq 2^{-\lambda}
\end{aligned}$$

Lastly $\Pr[E_S] = (1 - \Pr[\sum_{i=1}^{c} X_i < c_1]) \geq 1 - 2^{-\lambda}$.
□

C.3 Proof of Lemma 8

Proof. Lemma 6 requires $\varepsilon \geq 4(n' - d' + 1)/n'$ and yields $\varepsilon' = \varepsilon/4$. Lemma 7, on the other hand, require $\varepsilon' \geq 2\rho$ which implies $\varepsilon \geq 8\rho$. This proves the first part of the lemma. The second part follows directly from the setting of parameters for instantiation 1: namely $\rho = 2^{-\lambda}$ and $n' - d' + 1 = k' = t$, and $n' = 2^{\lambda}$.
□

C.4 Proof of Lemma 9

Proof. Let $\widetilde{\varepsilon} = \frac{\varepsilon}{4}$ and $\widetilde{n} = 2\frac{k'}{\widetilde{\varepsilon}^2} = \frac{32k'}{\varepsilon^2}$ so that $\widetilde{\varepsilon} \geq \sqrt{2k'/\widetilde{n}}$. On a query $e_1 \in [M]$, the machine $\mathcal{D}_2^{\mathcal{O}_F}$ works as follows:

1. Set $Q := \emptyset$.
2. For $i = 1, \ldots, \widetilde{n}$:
   Choose $e_2^{(i)} \leftarrow_R [n']$. If some value of the form $(e_2^{(i)}, \cdot)$ is in Q then quit.
   Else, let $\mu = \mathcal{O}_F((e_1, e_2^{(i)}))$ and set $Q := Q \cup \{(e_2^{(i)}, \mu)\}$.
3. Use the list-decoding algorithm on the set Q to recover a list of size $L = 2/\widetilde{\varepsilon}^2$ of candidates for the source value whose secondary encoding agrees with $\widetilde{\varepsilon}\widetilde{n}$ of the positions in Q. If list-decoding fails then quit [8].
4. Choose a uniformly random element from the list and return it.

Let us now analyze $\Pr_{e_1 \in [M]}\left[\mathcal{D}_2^{\mathcal{O}_F}(e_1) = C'[e_1]\right]$. As before, we call a value $\mathbf{e_1} \in [M]$ "*good*" if $\Pr_{e_2 \in [n']}\left[\mathcal{O}_F(\mathbf{e_1}, e_2) = C[(\mathbf{e_1}, e_2)]\right] \geq \varepsilon/2$. Let us define several events and random variables to aid us with the analysis:

- Let E_G be the event that the value e_1 is good.
- Let X_i be an indicator variable which is 1 if the answer received in iteration i is correct (matches $C[i]$) *or* if the algorithm quit before iteration i.
- Let E_Q be the event that the algorithm quits.
- Let E_L be the event that the algorithm recovers a list containining the correct value $C'[i]$.
- Let E_S be the event that the algorithm successfully outputs the correct value $C'[i]$.

As before, $\Pr[E_G] \geq \varepsilon/2$. Also, for ease of exposition, we use the notation $\Pr_G[\cdot]$ to denote the conditional probability $\Pr[\cdot \mid E_G]$. Firstly, although the events X_i are not independent (the probability of the algorithm quitting increases as Q grows), the probability of $X_i = 1$ is at least $p = \varepsilon/2$ conditioned on any values of $X_1, \ldots, X_{i-1}$. Reusing our analysis from the erasure case, the Chernoff-Hoeffding bound gives us:

$$\begin{aligned}\Pr_G\left[\sum_{i=1}^{\widetilde{n}} X_i < \widetilde{\varepsilon}\widetilde{n}\right] &= \Pr_G\left[\sum_{i=1}^{c} X_i < (1 - 1/2)p\widetilde{n}\right] \\ &\leq e^{-\frac{1}{8}(p\widetilde{n})} \leq e^{-\frac{\varepsilon}{16}\widetilde{n}} \leq e^{-1} \leq 1/2\end{aligned}$$

[8] An efficient algorithm for this task exists by Theorem 17. Think of the set Q as a $[\widetilde{n}, k', \widetilde{n} - k' + 1]_{q_r}$ Reed-Solomon code.

Now the event E_Q happens if on some iterations i, the value of $e_2^{(i)}$ is already in Q. On each iteration i, the size of Q is bounded by $\widetilde{n}-1$ and hence

$$\Pr_G[E_Q] \leq \sum_{1=1}^{\widetilde{n}} \frac{\widetilde{n}-1}{n'} \leq \frac{\widetilde{n}^2}{n'} \leq \frac{(32k'/\varepsilon^2)^2}{n'} \leq \frac{(32k')^2}{n'}\frac{1}{\varepsilon^4} \leq \frac{1}{8k'} \leq \frac{(32k')^2}{(8k')^4} \leq \frac{1}{4}.$$

Finally,

$$\begin{aligned}
\Pr_G[E_L] &= \Pr_G\left[\neg(\sum_{i=1}^{\widetilde{n}} X_i < \widetilde{\varepsilon}\widetilde{n}) \wedge \neg E_Q\right] \geq 1-1/2-1/4 \geq \frac{1}{4} \\
\Pr_G[E_S] &\geq (1/L)\Pr[E_L] \geq \frac{\widetilde{\varepsilon}^2}{8} \\
\Pr[E_S] &= \Pr_G[E_S]\Pr[E_G] \geq \Pr_G[E_S]\frac{\varepsilon}{2} \geq \frac{\widetilde{\varepsilon}^2\varepsilon}{16} \geq \frac{\varepsilon^3}{256}
\end{aligned}$$

□

C.5 Proof of Lemma 11

Proof. Starting with an oracle O_F which is an ε-error oracle for the final codeword C, we use Lemma 9 to (efficiently) convert it into an ε'-error oracle $\widetilde{O}_F$ for the codeword C', where $\varepsilon' = \varepsilon^3/256$. This requires $\varepsilon \geq 8\frac{k'}{(n')^{1/4}}$. In our instantiations, $n' = 2^\lambda$ and $k' = t \leq \widetilde{k}$. Therefore we can set $\varepsilon_0 = \frac{8\widetilde{k}}{2^{\lambda/4}}$.

Then, using Theorem 10, we efficiently convert $\widetilde{O}_F$ into L oracle functions $g_1, \ldots, g_L$ such that at least one of these functions is a δ-error oracle for the codeword F', where $\delta = 1-\mathcal{O}(\log\left(\frac{1}{\varepsilon'}\right)/t) = 1 - c\lambda/t$ for some constant c. Let us set $t \geq \frac{c\lambda}{\frac{1}{2}(1-1/\gamma)} = \mathcal{O}(\frac{\lambda}{\gamma-1})$ so that, recalling $\gamma \geq k/n$, we get $\delta \geq 1/2(1+\gamma^{-1}) \geq 1/2(1+k/n)$.

Now we can form L candidate codewords $F'_1, \ldots, F'_L$ by setting

$$F'_i = \left(g_i^{\widetilde{O}_F}(1), \ldots, g_i^{\widetilde{O}_F}(n)\right)$$

By Theorem 10, at least one of these has a $\delta \geq 1/2(1+k/n)$-fraction of positions in common with F'. By Theorem 18, for each F'_i we can efficiently find the unique codeword that agrees with a δ-fraction of the positions in F'_i. Hence, we recover a list of $L = \mathcal{O}(1/\varepsilon')$ possibilities, one of which includes F. □

D Proofs for Efficient PoR Scheme Constructions

D.1 Proof of Theorem 12

Proof. The efficiency parameters are obvious, so we go straight to the extractor $\mathcal{E}$ which is constructed from the error-decoder $\mathcal{D}$ for the underlying PoR code. In the rest of the proof we assume that $\varepsilon_0 \geq 2^{-\lambda}$ (a lower ε_0 does not improve the final soundness error ν).

The extractor: The extractor $\mathcal{E}^{\widetilde{P}}(\mathsf{ver}, k, \lambda)$ works as follows. It runs in iterations $i = 1, 2, 3, \ldots, \lambda$ and, on each iteration i it sets $\varepsilon = 2^{-i}$ and runs $\mathcal{D}^{\widetilde{P}}(k, \lambda, \varepsilon)$. If $\mathcal{D}$ outputs a list of candidates for F, than $\mathcal{E}$ scans through this list, outputs the first element $\bar{F}$ in the list such that $h(\bar{F}) = \omega$ and halts. Otherwise, if no such element exists or no list is produced, than $\mathcal{E}$ goes to the next iteration.
Soundness Error: Assume that the experiment $(v, w, T) \leftarrow \mathbf{Sound}^{\mathcal{E}}_{\mathcal{A}}(p_1(\lambda), p_2(\lambda))$ is run and let $\mathbf{G}$ be the event that the prover $\widetilde{P}$ chosen by the adversary $\mathcal{A}$ in this experiment has the property

$$\Pr_{e \leftarrow_R [N]}[\widetilde{P}(e) \text{ is correct }] \geq \varepsilon_0.$$

Then

$$\Pr[v = 1 \ \wedge \ w = 0] \ = \ \Pr[v = 1 \ \wedge \ w = 0 | \mathbf{G}] \Pr[\mathbf{G}] + \Pr[v = 1 \ \wedge \ w = 0 | \neg \mathbf{G}](1 - \Pr[\mathbf{G}])$$

We know that $\Pr[v = 1 \ \wedge \ w = 0 | \neg \mathbf{G}] \leq \Pr[v = 1 | \neg \mathbf{G}] \leq \varepsilon_0$ since the final challenge which determines $v \in \{0, 1\}$, is completely random and independent of the way that $\widetilde{P}$ is chosen and so it will be answered correctly with probability at most ε_0. Now we need to upper bound $\Pr[v = 1 \ \wedge \ w = 0 | \mathbf{G}] \leq \Pr[w = 0 | \mathbf{G}]$. Essentially we know that $\widetilde{P}$ is a ε_0 adversary so that $\mathcal{D}$ should succeeds in outputting a list containing F in iteration at most $i = \lambda$ with probability at least $(1 - 2^{-\lambda})$. So the only way that $\mathcal{E}$ does not succeed is if either (1) $\mathcal{D}$ never outputs a list containing F or (2) some other element $\bar{F} \neq F$ output by $\mathcal{D}$ in one of the iterations $1, \ldots, \lambda$ has the property that $h(\bar{F}) = \omega$. So

$$\Pr[v = 1 \ \wedge \ w = 0 | \mathbf{G}] \leq \Pr[\text{event (1)}] + \Pr[\text{event (2)}] \leq 2^{-\lambda} + \sum_{i=1}^{\lambda} a(\lambda 2^i)^b \psi \leq 2a(\lambda 2^\lambda)^b \psi. \quad (5)$$

Putting this together, we get

$$\Pr[v = 1 \ \wedge \ w = 0] \leq \varepsilon_0 \Pr[\mathbf{G}] + (2a(\lambda 2^\lambda)^b \psi + 2^{-\lambda})(1 - \Pr[\mathbf{G}]) \leq \max(\varepsilon_0, 2a(\lambda 2^\lambda)^b \psi + 2^{-\lambda})$$

as we wanted to show.
Knowledge Soundness: We are left to show that if the PoR code is efficiently error-decodable than the extractor is efficient. By assumption, if the PoR code is efficient, than the run-time of the decoder $\mathcal{D}(k, \lambda, \varepsilon)$ is bounded by $\mathsf{poly}(k, \lambda, 1/\varepsilon) \leq a'(\frac{k\lambda}{\varepsilon})^{b'}$ for some constants $a', b' \geq 1$. Therefore, if the extractor halts in iteration j, then its run time T is bounded by $\sum_{i=1}^{j} a'(k\lambda 2^i)^{b'} \leq 2a'(k\lambda 2^j)^{b'}$. Setting $c = 1/b'$, we see that, if the extractor halts in iteration j, then $T^c \leq a'' k\lambda 2^j$ for some constant a'', and overall $T^c \leq a''(k\lambda 2^\lambda)$.

Now let us consider the random experiment of the soundness game $(v, w, T) \leftarrow \mathbf{Sound}^{\mathcal{E}}_{\mathcal{A}}(p_1(\lambda), p_2(\lambda))$. Firstly, for any $i \geq 1$,

$$\Pr[T^c \geq i] \leq \Pr[v = 1, j \geq \log(i/(a'' k\lambda))] \quad (6)$$

where j is a random variable which indicates the iteration in which $\mathcal{E}$ halts. Let us define the event $\mathbf{G}$ to be the event that the prover $\widetilde{P}$ output by the adversary $\mathcal{A}$ during the soundness game is an $\varepsilon = a'' k\lambda / i$ error-oracle. Then

$$\begin{aligned} \Pr[T^c \geq i] \ &\leq \ \Pr[v = 1, j \geq \log(i/(a'' k\lambda)), \mathbf{G}] + \Pr[v = 1, j \geq \log(i/(a'' k\lambda), \neg \mathbf{G}] \quad (7) \\ &\leq \ 2^{-\lambda} + a'' k\lambda / i \quad (8) \end{aligned}$$

since the first part of (7) is bounded by the probability that $\mathcal{D}^{\widetilde{P}}(k,\lambda,\varepsilon)$ fails to output a list containing F when $\widetilde{P}$ is an ε adversary (which is at most $2^{-\lambda}$) and the second part of (7) is bounded by the probability that $\widetilde{P}$ succeeds in a proof and is not an ε-error adversary, which is bounded by ε. Therefore

$$\begin{aligned}\mathbf{E}[T^c] &\leq \sum_{i=1}^{\infty}\Pr[T^c \geq i] \leq \sum_{i=1}^{a''k\lambda 2^{\lambda}}(2^{-\lambda}+a''k\lambda/i)\\ &\leq a''k\lambda(1+\mathcal{O}(\log(a''k\lambda))) = \mathsf{poly}(k,\lambda)\end{aligned}$$

□

D.2 Proof of Theorem 13.

Proof. The proof proceeds similarly to that of Theorem 12 with only minor modifications.
The extractor: The extractor $\mathcal{E}$ is defined the same way as in the proof of Theorem 12, but it runs for a maximum of $-\log(\chi(\lambda))$ iterations where $\chi(\lambda)$ is a negligible function which bounds the success probability that the adversary $\mathcal{A}$ (which is a PPT adversary) distinguishes the PRFs $\mathcal{F}_1, F_2$ from random functions. We set $\varepsilon_0' = \max(\varepsilon_0, \chi, 2^{-\lambda})$.
Soundness: We analyze soundness by defining the event $\mathbf{G}$ the same way as in the proof of Theorem 12, but using ε_0' instead of ε_0. We get

$$\Pr[v=1 \ \wedge \ w=0] \quad = \quad \Pr[v=1 \ \wedge \ w=0,\mathbf{G}] + \Pr[v=1 \ \wedge \ w=0,\neg\mathbf{G}] = \mathrm{negl}(\lambda)$$

The analysis showing $\Pr[v=1 \ \wedge \ w=0,\mathbf{G}] = \mathrm{negl}(\lambda)$ is the same as before. In particular, if the prover $\widetilde{P}$ is an ε_0'-adversary than the extractor will succeed in outputting F with overwhelming probability. As the only modification, we will need to set the constant s depending on the specific polynomial bounds for $L(\lambda,\varepsilon)$ so that (looking at (5) in the proof of Theorem 12) the vale $2a(\lambda 2^{\lambda})^b\psi$ is negligible.

However, we now need modified analysis to show that $\Pr[v=1 \ \wedge \ w=0,\neg\mathbf{G}] = \mathrm{negl}(\lambda)$. Essentially, we must bound the probability that $\mathcal{A}$ produces a prover which answers a negligible (fewer than ε_0) portion of the challenges, but nevertheless succeeds in the "next" audit. We note that if the function families $\mathcal{F}_1, \mathcal{F}_2$ are random functions then it is easy to see that $\Pr[v=1 \ \wedge \ w=0,\neg\mathbf{G}] = \mathrm{negl}(\lambda)$, since the challenge sent in audit i is now completely random and independent of what the adversary sees, and the success probability of $\widetilde{P}$ succeeding is at most its probability of guessing $f_2(\ell,\mu')$ for some new μ', which is negligible. Therefore, when $\mathcal{F}_1, \mathcal{F}_2$ are PRF families, then $\Pr[v=1 \ \wedge \ w=0,\neg\mathbf{G}] = \mathrm{negl}(\lambda)$ since otherwise $\mathcal{A}$ could be used as a distinguisher. This concludes the analysis of soundness.
Knowledge Soundness: Assuming that the decoder $\mathcal{D}$ is efficient, we must show that the extractor $\mathbf{E}$ is also efficient. Again the analysis follows the proof of Theorem 12 with small modifications. Firstly, because we only run for at most $i = -\log(\chi(\lambda))$ iterations, we get the improved upper bound $T^c \leq a''\frac{k\lambda}{\chi(\lambda)}$ overall. Secondly, equations (7) and (8) now become

$$\begin{aligned}\Pr[T^c \geq i] &\leq \Pr[v=1, j \geq \log(i/(a''k\lambda)),\mathbf{G}] + \Pr[v=1, j \geq \log(i/(a''k\lambda),\neg\mathbf{G}] &(9)\\ &\leq 2^{-\lambda} + a''k\lambda/i + \chi(\lambda) &(10)\end{aligned}$$

where the bound $\Pr[v=1, j \geq \log(i/(a''k\lambda)),\mathbf{G}] \leq 2^{-\lambda}$ is the same as before (relying on the fact that the extractor halts in iteration j when given an $\varepsilon \geq 2^{-j}$ error-adversary with probability $2^{-\lambda}$).

However, $\Pr[v = 1, j \geq \log(i/(a''k\lambda), \neg\mathbf{G}]$ can only be shown to be equal to $a''k\lambda/i$ when $\mathcal{F}_1, \mathcal{F}_2$ are random functions. When they are PRFs we need to add in the probability of $\mathcal{A}$ distinguishing these from random, and thus we incur a factor of $\chi(\lambda)$. Therefore

$$\mathbf{E}[T^c] \leq \sum_{i=1}^{\infty} \Pr[T^c \geq i] \leq \sum_{i=1}^{\frac{a''k\lambda}{\chi(\lambda)}} (2^{-\lambda} + a''k\lambda/i + \chi(\lambda)) = \mathsf{poly}(k, \lambda)$$

□

E Homomorphic Linear Authenticators

We briefly review the linear authenticator construction of [SW08] for the symmetric key setting (see [SW08] for more details and also a public key based construction). Let $\mathcal{F} = \{f_k\}_{k \in \mathcal{K}_{\mathsf{prf}}}$ be a PRF family with outputs in $\mathbb{F}$. Then the linear authenticator construction is given is as follows:

Key Selection: Choose $K = (k, \alpha)$ uniformly at random from $\mathcal{K}_{prf} \times \mathbb{F}$.

Tagging $\mathsf{LinTag}_K(\overline{x})$**:** Let $\overline{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$. For each $i \in [1, \ldots, n]$ compute $\sigma_i = f_k(i) + \alpha x_i$ and output the vector-tag $\overline{\sigma} = (\sigma_1, \ldots, \sigma_n)$.

Authenticating $\mathsf{LinAuth}(\overline{x}, \overline{a}, \overline{\sigma})$**:** Compute $\mu = \sum_{i=1}^{n} a_i x_i$, $\sigma = \sum_{i=1}^{n} a_i \sigma_i$ and output (μ, σ). Note that this can be done by only reading values x_i, σ_i for which $a_i \neq 0$.

Verifying $\mathsf{LinVer}_K(\overline{a}, \mu', \sigma')$**:** Verify that $\sigma' = \alpha\mu' + \sum_{i=1}^{n} a_i f_k(i)$.

On an intuitive level, if an adversary can come up with $\mu' \neq \mu, \sigma'$ such that $\mathsf{LinVer}_K(\overline{a}, \mu', \sigma')$ verifies, then the adversary can also solve for $\alpha = (\sigma - \sigma')/(\mu - \mu')$ where μ, σ are the correct responses for the coefficients $\overline{a}$. However, α should be computationally hidden since it is *masked* by the values $f_k(i)$. See [SW08] for a formal proof.