Compact Proofs of Retrievability

Hovav Shacham[1] and Brent Waters[2⋆]

[1] University of California, San Diego
hovav@cs.ucsd.edu
[2] University of Texas, Austin
bwaters@csl.sri.com

Abstract. In a proof-of-retrievability system, a data storage center convinces a verifier that he is actually storing all of a client's data. The central challenge is to build systems that are both efficient and *provably* secure – that is, it should be possible to extract the client's data from any prover that passes a verification check. In this paper, we give the first proof-of-retrievability schemes with full proofs of security against *arbitrary* adversaries in the strongest model, that of Juels and Kaliski. Our first scheme, built from BLS signatures and secure in the random oracle model, has the *shortest query and response* of any proof-of-retrievability with public verifiability. Our second scheme, which builds elegantly on pseudorandom functions (PRFs) and is secure in the standard model, has the *shortest response* of any proof-of-retrievability scheme with private verifiability (but a longer query). Both schemes rely on homomorphic properties to aggregate a proof into one small authenticator value.

1 Introduction

In this paper, we give the first proof-of-retrievability schemes with full proofs of security against *arbitrary* adversaries in the Juels-Kaliski model. Our first scheme has the shortest query and response of any proof-of-retrievability with public verifiability and is secure in the random oracle model. Our second scheme has the shortest response of any proof-of-retrievability scheme with private verifiability (but a longer query), and is secure in the standard model.

Proofs of storage. Recent visions of "cloud computing" and "software as a service" call for data, both personal and business, to be stored by third parties, but deployment has lagged. Users of outsourced storage are at the mercy of their storage providers for the continued availability of their data. Even Amazon's S3, the best-known storage service, has recently experienced significant downtime.[3]

In an attempt to aid the deployment of outsourced storage, cryptographers have designed systems that would allow users to verify that their data is still

⋆ Supported by NSF CNS-0749931, CNS-0524252, CNS-0716199; the US Army Research Office under the CyberTA Grant No. W911NF-06-1-0316; and the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001.

[3] See, e.g., http://blogs.zdnet.com/projectfailures/?p=602.

available and ready for retrieval if needed: Deswarte, Quisquater, and Saïdane [8], Filho and Barreto [9], and Schwarz and Miller [15]. In these systems, the client and server engage in a protocol; the client seeks to be convinced by the protocol interaction that his file is being stored. Such a capability can be important to storage providers as well. Users may be reluctant to entrust their data to an unknown startup; an auditing mechanism can reassure them that their data is indeed still available.

Evaluation: formal security models. Such proof-of-storage systems should be evaluated by both "systems" and "crypto" criteria. Systems criteria include: (1) the system should be as *efficient* as possible in terms of both computational complexity and communication complexity of the proof-of-storage protocol, and the storage overhead on the server should be as small as possible; (2) the system should allow *unbounded use* rather than imposing a priori bound on the number of audit-protocol interactions[4]; (3) verifiers should be *stateless*, and not need to maintain and update state between audits, since such state is difficult to maintain if the verifier's machine crashes or if the verifier's role is delegated to third parties or distributed among multiple machine.[5] Statelessness and unbounded use are required for proof-of-storage systems with *public verifiability*, in which anyone can undertake the role of verifier in the proof-of-storage protocol, not just the user who originally stored the file.[6]

The most important crypto criterion is this: Whether the protocol actually establishes that any server that passes a verification check for a file – even a malicious server that exhibits arbitrary, Byzantine behavior – is *actually storing the file*. The early cryptographic papers lacked a formal security model, let alone proofs. But provable security matters. Even reasonable-looking protocols could in fact be insecure; see Appendix C of the full paper [16] for an example.

The first papers to consider formal models for proofs of storage were by Naor and Rothblum, for "authenticators" [14], and by Juels and Kaliski, for "proofs of retrievability" [12]. Though the details of the two models are different, the insight behind both is the same: in a secure system if a server can pass an audit then a special extractor algorithm, interacting with the server, must be able (w.h.p.) to extract the file.[7]

[4] We believe that systems allowing a bounded number of interactions can be useful, but only as stepping stones towards fully secure systems. Some examples are bounded identity-based encryption [11] and bounded CCA-secure encryption [7]; in these systems, security is maintained only as long as the adversary makes at most t private key extraction or decryption queries.

[5] We note that the sentinel-based scheme of Juels and Kaliski [12], the scheme of Ateniese, Di Pietro, Mancini, and Tsudik [3], and the scheme of Shah, Swaminathan and Baker [17] lack both unbounded use and statelessness. We do not consider these schemes further in this paper.

[6] Ateniese et al. [1] were the first to consider public verifiability for proof-of-storage schemes.

[7] This is, of course, similar to the intuition behind proofs of knowledge.

A simple MAC-based construction. In addition, the Naor-Rothblum and Juels-Kaliski papers describe similar proof-of-retrievability protocols. The insight behind both is that checking that *most* of a file is stored is easier than checking that *all* is. If the file to be stored is first encoded redundantly, and each block of the encoded file is authenticated using a MAC, then it is sufficient for the client to retrieve retrieves a few blocks together with their MACs and check, using his secret key, that these blocks are correct. Naor and Rothblum prove their scheme secure in their model.[8] The simple protocol obtained here uses techniques similar to those proposed by Lillibridge et al. [13]. Signatures can be used instead of MACs to obtain public verifiability.

The downside to this simple solution is that the server's response consists of λ block-authenticator pairs, where λ is the security parameter. If each authenticator is λ bits long, as required in the Juels-Kaliski model, then the response is $\lambda^2 \cdot (s+1)$ bits, where the ratio of file block to authenticator length is $s : 1$.[9]

Homomorphic authenticators. The proof-of-storage scheme described by Ateniese et al. [1] improves on the response length of the simple MAC-based scheme using *homomorphic authenticators*. In their scheme, the authenticators σ_i on each file block m_i are constructed in such a way that a verifier can be convinced that a linear combination of blocks $\sum_i \nu_i m_i$ (with arbitrary weights $\{\nu_i\}$) was correctly generated using an authenticator computed from $\{\sigma_i\}$.[10]

When using homomorphic authenticators, the server can combine the blocks and λ authenticators in its response into a single aggregate block and authenticator, reducing the response length by a factor of λ. As an additional benefit, the Ateniese et al. scheme is the first with public verifiability. The homomorphic authenticators of Ateniese et al. are based on RSA and are thus relatively long.

Unfortunately, Ateniese et al. do not give a rigorous proof of security for their scheme. In particular, they do not show that one can extract a file (or even a significant fraction of one) from a prover that is able to answer auditing queries convincingly. The need for rigor in extraction arguments applies equally to both the proof-of-retrievability model we consider and the weaker proof of data possession model considered by Ateniese et al.[11]

Our contributions. In this paper, we make two contributions.

1. We describe two new short, efficient homomorphic authenticators. The first, based on PRFs, gives a proof-of-retrievability scheme secure in the stan-

[8] Juels and Kaliski do not give a proof of security against arbitrary adversaries, but this proof is trivial using the techniques we develop in this paper; for completeness, we give the proof in Appendix D of the full paper [16].

[9] Naor and Rothblum show that one-bit MACs suffice for proving security in their less stringent model, for an overall response length of $\lambda \cdot (s+1)$ bits. The Naor-Rothblum scheme is not secure in the Juels-Kaliski model.

[10] In the Ateniese et al. construction the aggregate authenticator is $\prod_i \sigma_i^{\nu_i} \bmod N$.

[11] For completeness, we give a correct and fully proven Ateniese-et-al.–inspired, RSA-based scheme, together with a full proof of security, in Appendix E of the full paper [16].

dard model. The second, based on BLS signatures [5], gives a proof-of-retrievability scheme with public verifiability secure in the random oracle model.
2. We prove both of the resulting schemes secure in a variant of the Juels-Kaliski model. Our schemes are the first with a security proof against arbitrary adversaries in this model.

The scheme with public retrievability has the shortest query and response of any proof-of-retrievability scheme: 20 bytes and 40 bytes, respectively, at the 80-bit security level. The scheme with private retrievability has the shortest response of any proof-of-retrievability scheme (20 bytes), matching the response length of the Naor-Rothblum scheme in a more stringent security model, albeit at the cost of a longer query. We believe that derandomizing the query in this scheme is the major remaining open problem for proofs of retrievability.

1.1 Our Schemes

In our schemes, as in the Juels-Kaliski scheme, the user breaks an erasure encoded file into n blocks $m_1, \ldots, m_n \in Z_p$ for some large prime p. The erasure code should allow decoding in the presence of *adversarial* erasure. Erasure codes derived from Reed-Solomon codes have this property, but decoding and encoding are slow for large files. In Appendix B of the full paper [16] we discuss how to make use of more efficient codes secure only against random erasures.

The user authenticates each block as follows. She chooses a random $\alpha \in \mathbb{Z}_p$ and PRF key k for function f. These values serve as her secret key. She calculates an authentication value for each block i as

$$\sigma_i = f_k(i) + \alpha m_i \in \mathbb{Z}_p \ .$$

The blocks $\{m_i\}$ and authenticators $\{\sigma_i\}$ are stored on the server. The proof of retrievability protocol is as follows. The verifier chooses a random challenge set I of l indices along with l random coefficients in Z_p.[12] Let Q be the set $\{(i, \nu_i)\}$ of challenge index–coefficient pairs. The verifier sends Q to the prover. The prover then calculates the response, a pair (σ, μ), as

$$\sigma \leftarrow \sum_{(i,\nu_i)\in Q} \nu_i \cdot \sigma_i \qquad \text{and} \qquad \mu \leftarrow \sum_{(i,\nu_i)\in Q} \nu_i \cdot m_i \ .$$

Now verifier can check that the response was correctly formed by checking that

$$\sigma \stackrel{?}{=} \alpha \cdot \mu + \sum_{(i,\nu_i)\in Q} \nu_i \cdot f_k(i) \ .$$

It is clear that our techniques admit short responses. But it is not clear that our new system admits a simulator that can extract files. Proving that it does is quite challenging, as we discuss below. In fact, unlike similar, seemingly correct schemes (see Appendix C of the full paper [16]), our scheme is provably secure in the standard model.

[12] Or, more generally, from a subset B of $\mathbb{Z}_p$ of appropriate size; see Section 1.1.

A scheme with public verifiability. Our second scheme is publicly verifiable. It follows the same framework as the first, but instead uses BLS signatures [5] for authentication values that can be publicly verified. The structure of these signatures allows for them to be aggregated into linear combinations as above. We prove the security of this scheme under the Computational Diffie-Hellman assumption over bilinear groups in the random oracle model.

Let $e\colon G \times G \to G_T$ be a computable bilinear map with group G's support being $\mathbb{Z}_p$. A user's private key is $x \in \mathbb{Z}_p$, and her public key is $v = g^x \in G$ along with another generator $u \in G$. The signature on block i is $\sigma_i = \big[H(i)u^{m_i}\big]^x$. On receiving query $Q = \{(i, \nu_i)\}$, the prover computes and sends back $\sigma \leftarrow \prod_{(i,\nu_i)\in Q} \sigma_i^{\nu_i}$ and $\mu \leftarrow \sum_{(i,\nu_i)\in Q} \nu_i \cdot m_i$. The verification equation is:

$$e(\sigma, g) \stackrel{?}{=} e\Big(\prod_{(i,\nu_i)\in Q} H(i)^{\nu_i} \cdot u^{\mu},\ v\Big) \ .$$

This scheme has public verifiability: the private key x is required for generating the authenticators $\{\sigma_i\}$ but the *public* key v is sufficient for the verifier in the proof-of-retrievability protocol.

Parameter selection. Let λ be the security parameter; typically, $\lambda = 80$. For the scheme with private verification, p should be a λ bit prime. For the scheme with public verification, p should be a 2λ-bit prime, and the curve should be chosen so that discrete logarithm is 2^λ-secure. For values of λ up to 128, Barreto-Naehrig curves [4] are the right choice; see the survey by Freeman, Scott, and Teske [10].

Let n be the number of blocks in the file. We assume that $n \gg \lambda$. Suppose we use a rate-ρ erasure code, i.e., one in which any ρ-fraction of the blocks suffices for decoding. (Encoding will cause the file length to grow approximately $(1/\rho)\times$.) Let l be the number of indices in the query Q, and $B \subseteq \mathbb{Z}_p$ be the set from which the challenge weights ν_i are drawn.

Our proofs – see Section 4.2 for the details – guarantee that extraction will succeed from any adversary that convincingly answers an ϵ-fraction of queries, provided that $\epsilon - \rho^l - 1/\#B$ is non-negligible in λ. It is this requirement that guides the choice of parameters.

A conservative choice is $\rho = 1/2$, $l = \lambda$, and $B = \{0,1\}^\lambda$; this guarantees extraction against any adversary.[13] For applications that can tolerate a larger error rate these parameters can be reduced. For example, if a 1-in-1,000,000 error is acceptable, we can take B to be the set of 22-bit strings and l to be 22; alternatively, the coding expansion $1/\rho$ can be reduced.

A tradeoff between storage and communication. As we described our schemes above, each file block is accompanied by an authenticator of equal length. This

[13] The careful analysis in our proofs allows us to show that, for 80-bit security, the challenge coefficients ν_i can be 80 bits long, not 160 as proposed in [2, p. 17]. The smaller these coefficients, the more efficient the multiplications or exponentiations that involve them.

gives a $2\times$ overhead beyond that imposed by the erasure code, and the server's response in the proof-of-retrievability protocol is $2\times$ the length of an authenticator. In the full schemes of Section 3, we introduce a parameter s that gives a tradeoff between storage overhead and response length. Each block consists of s elements of $\mathbb{Z}_p$ that we call *sectors*. There is one authenticator per block, reducing the overhead to $(1 + 1/s)\times$. The server's response is one aggregated block and authenticator, and is $(1 + s)\times$ as long as an authenticator. The choice $s = 1$ corresponds to our schemes as we described them above and to the scheme given by Ateniese et al. [1].[14]

Compressing the request. A request, as we have seen, consists of an l element subset of $[1, n]$ together with l elements of the coefficient set B, chosen uniformly and independently at random. In the conservative parametrization above, a request is thus $\lambda \cdot \left(\lceil \lg n \rceil + \lambda\right)$ bits long. One can reduce the randomness required to *generate* the request using standard techniques,[15] but this will not shorten the request itself. In the random oracle model, the verifier can send a short (2λ bit) seed for the random oracle from which the prover will generate the full query. Using this technique we can make the queries as well as responses compact in our publicly verifiable scheme, which already relies on random oracles.[16] Obtaining short queries in the standard model is the major remaining open problem in proofs of retrievability.

We note that, by techniques similar to those discussed above, a PRF can be used to generate the per-file secret values $\{\alpha_j\}$ for our privately verifiable scheme and a random oracle seed can be used to generate the per-file public generators $\{u_j\}$ in our publicly verifiable scheme. This allows file tags for both schemes to be short: $O(\lambda)$, asymptotically.

We also note that subsequent to our work Bowers, Juels, and Oprea [6] provided a framework, based on "inner and outer" error correcting codes, by which they describe parameterizations of our approach that trade off the cost of a single audit and the computational efficiency of extracting a file a series of audit requests. In our work we have chosen to put emphasis on reducing single audit costs. We envision an audit as a mechanism to ensure that a file is indeed available and that a file under most circumstances will be retrieved as a simple bytestream. In a further difference, the error-correcting codes employed by Bowers, Juels, and Oprea are optimized for the case where $\epsilon > 1/2$, i.e., for

[14] It would be possible to shorten the response further using knowledge-of-exponent assumptions, as Ateniese et al. do, but such assumptions are strong and nonstandard; more importantly, their use means that the extractor can never be implemented in the real world.

[15] For example, choose keys k' and k'' for PRFs with respective ranges $[1, n]$ and B. The query indices are the first l distinct values amongst $f'_{k'}(1), f'_{k'}(2), \ldots$; the query coefficients are $f''_{k''}(1), \ldots, f''_{k''}(l)$.

[16] Ateniese et al. propose to eliminate random oracles here by having the prover generate the full query using PRF keys sent by the verifier [2, p. 11], but it is not clear how to prove such a scheme secure, since the PRF security definition assumes that keys are kept secret.

when the server answers correctly more than half the time. By contrast, our techniques scale to any small (but nonnegligible) ϵ. We believe that this frees systems implementers from having to worry about whether a substantial error rate (for example, due to an intermitent connection between auditor and server) invalidates the assumptions of the underlying cryptography.

1.2 Our Proofs

We provide a modular proof framework for the security of our schemes. Our framework allows us to argue about the systems unforgeability, extractability, and retrievability with these three parts based respectively on cryptographic, combinatorial, and coding-theoretical techniques. Only the first part differs between the three schemes we propose. The combinatorial techniques we develop are nontrivial and we believe they will be of independent interest.

It is interesting to compare both our security model and our proof methodology to those in related work.

The proof of retrievability model has two major distinctions from that used by Naor and Rothblum [14] (in addition to the public-key setting). First, the NR model assumes a checker can request and receive specific memory locations from the prover. In the proof of retrievability model, the prover can consist of an arbitrary program as opposed to a simple memory layout and this program may answer these questions in an arbitrary manner. We believe that this realistically represents an adversary in the type of setting we are considering. In the NR setting the extractor needs to retrieve the file given the server's memory; in the POR setting the analogy is that the extractor receives the adversary's program.

Second, in the proof of retrievability model we allow the attacker to execute a polynomial number of proof attempts before committing to how it will store memory. In the NR model the adversary does not get to execute the protocol before committing its memory. This weaker model is precisely what allows for the use of 1-bit MACs with error correcting codes in one NR variant. One might argue that in many situations this is sufficient. If a storage server responds incorrectly to an audit request we might assume that it is declared to be cheating and there is no need to go further. However, this limited view overlooks several scenarios. In particular, we want to be able to handle setups where there are several verifiers that do not communicate or if there might be several storage servers handling the same encoded file that are audited independently. Only our stronger model can correctly reflect these situations. In general, we believe that the strongest security model allows for a system to be secure in the most contexts including those not previously considered.[17]

One of the distinctive and challenging parts of our work is to argue extraction from homomorphically accumulated blocks. While Ateniese et al. [1] proposed using homomorphic RSA signatures and proved what is equivalent to our unforgeability requirement, they did not provide an argument that one could

[17] We liken this argument to that for the strong definition currently accepted for chosen-ciphertext secure encryption.

extract individual blocks from a prover. The only place where extractability is addressed in their work is a short paragraph in Appendix A, where they provide some intuitive arguments. Here is one concrete example: Their constructions make multiple uses of pseudorandom functions (PRFs), yet the security properties of a PRF are never applied in a security reduction. This gives compelling evidence that a rigorous security proof was not provided. Again, we emphasize that extraction is needed in even the weaker proof of data possession model claimed by the authors.

Extractability issues arise in several natural constructions. Proving extraction from aggregated authenticator values can be challenging; in Appendix C of the full paper [16] we show an attack on a natural but incorrect system that is very similar to the "E-PDP" efficient alternative scheme given by Ateniese et al. (which they use in their performance measurements). For this scheme, Ateniese et al. claim only that the protocol establishes that a cheating prover has the sum $\sum_{i \in I} m_i$ of the blocks. We show that indeed this is all it can provide. Ateniese et al. calculate that a malicious server attacking the E-PDP scheme et al. that a malicious server attacking the E-PDP scheme would need to store 10^{140} blocks in order to cheat with probability 100%. By contrast, our attack, which allows the server to cheat with somewhat lower probability (almost 9% for standard parameters) requires no more storage than were the server faithfully storing the file.

Finally, we argue that the POR is the "right" model for considering practical data storage problems, since provides a successful audit guarantees that *all* the data can be extracted. Other work has advocated that a weaker Proof of Data Possession [1] model might be acceptable. In this model, one only wants to guarantee that a certain percentage (e.g., 90%) of data blocks are available. By offering this weaker guarantee one might hope to avoid the overhead of applying erasure codes. However, this weaker condition is unsatisfactory for most practical application demands. One might consider how happy a user would be were 10% of a file containing accounting data lost. Or if, for a compressed file, the compression tables were lost – and with them all useful data. Instead of hoping that there is enough redundancy left to reconstruct important data in an ad-hoc way, it is much more desirable to have a model that inherently provides this. We also note that Ateniese et al. [1] make an even weaker guarantee for their "E-PDP" system that they implement and use as the basis for their measurements. According to [1] their E-PDP system "only guarantees possession of the sum of the blocks." While this might be technically correct, it is even more difficult to discern what direct use could come from retrieving a sum of a subset of data blocks.

One might still hope to make use of systems proved secure under these models. For example, we might attempt to make a PDP system usable by adding on an erasure encoding step. In addition, if a system proved that one could be guaranteed sums of blocks for a particular audit, then it *might* be the case that by using multiple audit one could guarantee that individual file blocks could be extracted. However, one must prove that this is the case and account for the additional computational and communication overhead of multiple passes.

When systems use definitions that don't model full retrievability it becomes very difficult to make any useful security or performance comparisons.

2 Security Model

We recall the security definition of Juels and Kaliski [12]. Our version differs from the original definition in several details:

- we rule out any state ("α") in key generation and in verification, because (as explained in Section 1) we believe that verifiers in proof-of-retrievability schemes should be stateless;
- we allow the proof protocol to be arbitrary, rather than two-move, challenge-response; and
- our key generation emits a public key as well as a private key, to allow us to capture the notion of public verifiability.

Note that any stateless scheme secure in the original Juels-Kaliski model will be secure in our variant, and any scheme secure in our variant whose proof protocol can be cast as two-move, challenge-response protocol will be secure in the Juels-Kaliski definition. In particular, our scheme with private verifiability is secure in the original Juels-Kaliski model.[18]

A proof of retrievability scheme defines four algorithms, Kg, St, $\mathcal{V}$, and $\mathcal{P}$, which behave thus:

$\mathsf{Kg}()$. This randomized algorithm generates a public-private keypair (pk, sk).

$\mathsf{St}(sk, M)$. This randomized file-storing algorithm takes a secret key sk and a file $M \in \{0,1\}^*$ to store. It processes M to produce and output M^*, which will be stored on the server, and a tag t. The tag contains information that names the file being stored; it could also contain additional secret information encrypted under the secret key sk.

$\mathcal{P}$, $\mathcal{V}$. The randomized proving and verifying algorithms define a protocol for proving file retrievability. During protocol execution, both algorithms take as input the public key pk and the file tag t output by St. The prover algorithm also takes as input the processed file description M^* that is output by St, and the verifier algorithm takes as input the secret key. At the end of the protocol run, $\mathcal{V}$ outputs 0 or 1, where 1 means that the file is being stored on the server. We can denote a run of two machines executing the algorithms as: $\{0,1\} \stackrel{\mathrm{R}}{\leftarrow} \big(\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{P}(pk, t, M^*)\big)$.

[18] In an additional minor difference, we do not specify the extraction algorithm as part of a scheme, because we do not expect that the extract algorithm will be deployed in outsourced storage applications. Nevertheless, the extract algorithm used in our proofs (cf. Section 4.2) is quite simple: undertake many random $\mathcal{V}$ interactions with the cheating prover; keep track of those queries for which $\mathcal{V}$ accepts the cheating prover's reply as valid; and continue until enough information has been gathered to recover file blocks by means of linear algebra. The adversary $\mathcal{A}$ could implement this algorithm by means of its proof-of-retrievability protocol access.

We would like a proof-of-retrievability protocol to be correct and sound. Correctness requires that, for all keypairs (pk, sk) output by Kg, for all files $M \in \{0,1\}^*$, and for all (M^*, t) output by $\mathsf{St}(sk, M)$, the verification algorithm accepts when interacting with the valid prover:

$$\big(\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{P}(pk, t, M^*)\big) = 1 \ .$$

A proof-of-retrievability protocol is sound if any cheating prover that convinces the verification algorithm that it is storing a file M is actually storing that file, which we define in saying that it yields up the file M to an extractor algorithm that interacts with it using the proof-of-retrievability protocol. We formalize the notion of an extractor and then give a precise definition for soundness.

An extractor algorithm $\mathsf{Extr}(pk, sk, t, \mathcal{P}')$ takes the public and private keys, the file tag t, and the description of a machine implementing the prover's role in the proof-of-retrievability protocol: for example, the description of an interactive Turing machine, or of a circuit in an appropriately augmented model. The algorithm's output is the file $M \in \{0,1\}^*$. Note that Extr is given non–black-box access to $\mathcal{P}'$ and can, in particular, rewind it.

Consider the following setup game between an adversary $\mathcal{A}$ and an environment:

1. The environment generates a keypair (pk, sk) by running Kg, and provides pk to $\mathcal{A}$.
2. The adversary can now interact with the environment. It can make queries to a store oracle, providing, for each query, some file M. The environment computes $(M^*, t) \xleftarrow{\mathrm{R}} \mathsf{St}(sk, M)$ and returns both M^* and t to the adversary.
3. For any M on which it previously made a store query, the adversary can undertake executions of the proof-of-retrievability protocol, by specifying the corresponding tag t. In these protocol executions, the environment plays the part of the verifier and the adversary plays the part of the prover: $\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{A}$. When a protocol execution completes, the adversary is provided with the output of $\mathcal{V}$. These protocol executions can be arbitrarily interleaved with each other and with the store queries described above.
4. Finally, the adversary outputs a challenge tag t returned from some store query, and the description of a prover $\mathcal{P}'$.

The cheating prover $\mathcal{P}'$ is *ϵ-admissible* if it convincingly answers an ϵ fraction of verification challenges, i.e., if $\Pr\big[\big(\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{P}'\big) = 1\big] \geq \epsilon$. Here the probability is over the coins of the verifier and the prover. Let M be the message input to the store query that returned the challenge tag t (along with a processed version M^* of M).

Definition 1. *We say a proof-of-retrievability scheme is ϵ-sound if there exists an extraction algorithm Extr such that, for every adversary $\mathcal{A}$, whenever $\mathcal{A}$, playing the setup game, outputs an ϵ-admissible cheating prover $\mathcal{P}'$ for a file M, the extraction algorithm recovers M from $\mathcal{P}'$ – i.e., $\mathsf{Extr}(pk, sk, t, \mathcal{P}') = M$ – except possibly with negligible probability.*

Note that it is okay for $\mathcal{A}$ to have engaged in the proof-of-retrievability protocol for M in its interaction with the environment. Note also that each run of the proof-of-retrievability protocol is independent: the verifier implemented by the environment is stateless.

Finally, note that we require that extraction succeed (with all but negligible probability) from an adversary that causes $\mathcal{V}$ to accept with any nonnegligible probability ϵ. An adversary that passes the verification even a very small but nonnegligible fraction of the time – say, once in a million interactions – is fair game. Intuitively, recovering enough blocks to reconstruct the original file from such an adversary should take $O(n/\epsilon)$ interactions; our proofs achieve essentially this bound.

Concrete or asymptotic formalization. A proof-of-retrievability scheme is secure if no efficient algorithm wins the game above except rarely, where the precise meaning of "efficient" and "rarely" depends on whether we employ a concrete of asymptotic formalization.

It is possible to formalize the notation above either concretely or asymptotically. In a concrete formalization, we require that each algorithm defining the proof-of-retrievability scheme run in at most some number of steps, and that for any algorithm $\mathcal{A}$ that runs in time t steps, that makes at most q_S store queries, and that undertakes at most q_P proof-of-retrievability protocol executions, extraction from an ϵ-admissible prover succeeds except with some small probability δ. In an asymptotic formalization, every algorithm is provided with an additional parameter 1^λ for security parameter λ, we require each algorithm to run in time polynomial in λ, and we require that extraction fail from an ϵ-admissible prover with only negligible probability in λ, provided ϵ is nonnegligible.

Public or private verification, public or private extraction. In the model above, the verifier and extractor are provided with a secret that is not known to the prover or other parties. This is a secret-verification, secret-extraction model model. If the verification algorithm does not use the secret key, any third party can check that a file is being stored, giving public verification. Similarly, if the extract algorithm does not use the secret key, any third party can extract the file from a server, giving public extraction.

3 Constructions

In this section we give formal descriptions for both our private and public verification systems. The systems here follow the constructions outlined in the introduction with a few added generalizations. First, we allow blocks to contain $s \geq 1$ elements of $\mathbb{Z}_p$. This allows for a tradeoff between storage overhead and communication overhead. Roughly the communication complexity grows as $s+1$ elements of $\mathbb{Z}_p$ and the ratio of authentication overhead to data stored (post encoding) is $1 : s$. Second, we describe our systems where the set of coefficients sampled from B can be smaller than all of $\mathbb{Z}_p$. This enables us to take advantage make more efficient systems in certain situations.

3.1 Common Notation

We will work in the group $\mathbb{Z}_p$. When we work in the bilinear setting, the group $\mathbb{Z}_p$ is the support of the bilinear group G, i.e., $\#G = p$. In queries, coefficients will come from a set $B \subseteq \mathbb{Z}_p$. For example, B could equal $\mathbb{Z}_p$, in which case query coefficients will be randomly chosen out of all of $\mathbb{Z}_p$.

After a file undergoes preliminary processing, the processed file is split into *blocks*, and each block is split into *sectors*. Each sector is one element of $\mathbb{Z}_p$, and there are s sectors per block. If the processed file is b bits long, then there are $n = \lceil b/s \lg p \rceil$ blocks. We will refer to individual file sectors as $\{m_{ij}\}$, with $1 \le i \le n$ and $1 \le j \le s$.

Queries. A query is an l-element set $Q = \{(i, \nu_i)\}$. Each entry $(i, \nu_i) \in Q$ is such that i is a block index in the range $[1, n]$, and ν_i is a multiplier in B. The size l of Q is a system parameter, as is the choice of the set B.

The verifier chooses a random query as follows. First, she chooses, uniformly at random, an l-element subset I of $[1, n]$. Then, for each element $i \in I$ she chooses, uniformly at random, an element $\nu_i \xleftarrow{R} B$. We observe that this procedure implies selection of l elements from $[1, n]$ *without* replacement but a selection of l elements from B *with* replacement.

Although the set notation $Q = \{(i, \nu_i)\}$ is space-efficient and convenient for implementation, we will also make use of a vector notation in the analysis. A query Q over indices $I \subset [1, n]$ is represented by a vector $\boldsymbol{q} \in (\mathbb{Z}_p)^n$ where $\boldsymbol{q}_i = \nu_i$ for $i \in I$ and $\boldsymbol{q}_i = 0$ for all $i \notin I$. Equivalently, letting $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ be the usual basis for $(\mathbb{Z}_p)^n$, we have $\boldsymbol{q} = \sum_{(i,\nu_i)\in Q} \nu_i \boldsymbol{u}_i$.[19]

If the set B does not contain 0 then a random query (according to the selection procedure defined above) is a random weight-l vector in $(\mathbb{Z}_p)^n$ with coefficients in B. If B does contain 0, then a similar argument can be made, but care must be taken to distinguish the case "$i \in I$ and $\nu_i = 0$" from the case "$i \notin I$."

Aggregation. For its response, the server responds to a query Q by computing, for each j, $1 \le j \le s$, the value

$$\mu_j \leftarrow \sum_{(i,\nu_i)\in Q} \nu_i m_{ij} \ .$$

That is, by combining sectorwise the blocks named in Q, each with its multiplier ν_i. Addition, of course, is modulo p. The response is $(\mu_1, \ldots, \mu_s) \in \left(\mathbb{Z}_p\right)^s$.

Suppose we view the message blocks on the server as an $n \times s$ element matrix $M = (m_{ij})$, then, using the vector notation for queries given above, the server's response is given by $\boldsymbol{q}M$.

[19] We are using subscripts to denote vector elements (for $\boldsymbol{q}$) and to choose a particular vector from a set (for $\boldsymbol{u}$); but no confusion should arise.

3.2 Construction for Private Verification

Let $f\colon \{0,1\}^* \times \mathcal{K}_{\text{prf}} \to \mathbb{Z}_p$ be a PRF.[20] The construction of the private verification scheme Priv is:

Priv.Kg(). Choose a random symmetric encryption key $k_{\text{enc}} \xleftarrow{\text{R}} \mathcal{K}_{\text{enc}}$ and a random MAC key $k_{\text{mac}} \xleftarrow{\text{R}} \mathcal{K}_{\text{mac}}$. The secret key is $sk = (k_{\text{enc}}, k_{\text{mac}})$; there is no public key.

Priv.St(sk, M). Given the file M, first apply the erasure code to obtain M'; then split M' into n blocks (for some n), each s sectors long: $\{m_{ij}\}_{\substack{1 \le i \le n \\ 1 \le j \le s}}$. Now choose a PRF key $k_{\text{prf}} \xleftarrow{\text{R}} \mathcal{K}_{\text{prf}}$ and s random numbers $\alpha_1, \ldots, \alpha_s \xleftarrow{\text{R}} \mathbb{Z}_p$. Let t_0 be $n \| \mathsf{Enc}_{k_{\text{enc}}}(k_{\text{prf}} \| \alpha_1 \| \cdots \| \alpha_s)$; the file tag is $t = t_0 \| \mathsf{MAC}_{k_{\text{mac}}}(t_0)$. Now, for each i, $1 \le i \le n$, compute

$$\sigma_i \leftarrow f_{k_{\text{prf}}}(i) + \sum_{j=1}^{s} \alpha_j m_{ij} \ .$$

The processed file M^* is $\{m_{ij}\}$, $1 \le i \le n$, $1 \le j \le s$ together with $\{\sigma_i\}$, $1 \le i \le n$.

Priv.$\mathcal{V}(pk, sk, t)$. Parse sk as $(k_{\text{enc}}, k_{\text{mac}})$. Use k_{mac} to verify the MAC on t; if the MAC is invalid, reject by emitting 0 and halting. Otherwise, parse t and use k_{enc} to decrypt the encrypted portions, recovering n, k_{prf}, and $\alpha_1, \ldots, \alpha_s$. Now pick a random l-element subset I of the set $[1, n]$, and, for each $i \in I$, a random element $\nu_i \xleftarrow{\text{R}} B$. Let Q be the set $\{(i, \nu_i)\}$. Send Q to the prover. Parse the prover's response to obtain $\mu_1, \ldots, \mu_s$ and σ, all in $\mathbb{Z}_p$. If parsing fails, fail by emitting 0 and halting. Otherwise, check whether

$$\sigma \stackrel{?}{=} \sum_{(i,\nu_i) \in Q} \nu_i f_{k_{\text{prf}}}(i) + \sum_{j=1}^{s} \alpha_j \mu_j \ ;$$

if so, output 1; otherwise, output 0.

Priv.$\mathcal{P}(pk, t, M^*)$. Parse the processed file M^* as $\{m_{ij}\}$, $1 \le i \le n$, $1 \le j \le s$, along with $\{\sigma_i\}$, $1 \le i \le n$. Parse the message sent by the verifier as Q, an l-element set $\{(i, \nu_i)\}$, with the i's distinct, each $i \in [1, n]$, and each $\nu_i \in B$. Compute

$$\mu_j \leftarrow \sum_{(i,\nu_i) \in Q} \nu_i m_{ij} \quad \text{for } 1 \le j \le s, \qquad \text{and} \qquad \sigma \leftarrow \sum_{(i,\nu_i) \in Q} \nu_i \sigma_i \ .$$

Send to the prover in response the values $\mu_1, \ldots, \mu_s$ and σ.

[20] In fact, the domain need only be $\lceil \lg N \rceil$-bit strings, where N is a bound on the number of blocks in a file.

3.3 Construction for Public Verification

Let $e\colon G \times G \to G_T$ be a bilinear map, let g be a generator of G, and let $H\colon \{0,1\}^* \to G$ be the BLS hash, treated as a random oracle.[21] The construction of the public verification scheme Pub is:

Pub.Kg(). Generate a random signing keypair $(spk, ssk) \xleftarrow{R} \mathsf{SKg}$. Choose a random $\alpha \xleftarrow{R} \mathbb{Z}_p$ and compute $v \leftarrow g^\alpha$. The secret key is $sk = (\alpha, ssk)$; the public key is $pk = (v, spk)$.

Pub.St(sk, M)**.** Given the file M, first apply the erasure code to obtain M'; then split M' into n blocks (for some n), each s sectors long: $\{m_{ij}\}_{\substack{1 \le i \le n \\ 1 \le j \le s}}$. Now parse sk as (α, ssk). Choose a random file name *name* from some sufficiently large domain (e.g., $\mathbb{Z}_p$). Choose s random elements $u_1, \ldots, u_s \xleftarrow{R} G$. Let t_0 be "$name \| n \| u_1 \| \cdots \| u_s$"; the file tag t is t_0 together with a signature on t_0 under private key ssk: $t \leftarrow t_0 \| \mathsf{SSig}_{ssk}(t_0)$. For each i, $1 \le i \le n$, compute

$$\sigma_i \leftarrow \Big(H(name \| i) \cdot \prod_{j=1}^{s} u_j^{m_{ij}} \Big)^{\alpha} \ .$$

The processed file M^* is $\{m_{ij}\}$, $1 \le i \le n$, $1 \le j \le s$ together with $\{\sigma_i\}$, $1 \le i \le n$.

Pub.$\mathcal{V}(pk, sk, t)$**.** Parse pk as (v, spk). Use spk to verify the signature on on t; if the signature is invalid, reject by emitting 0 and halting. Otherwise, parse t, recovering *name*, n, and $u_1, \ldots, u_s$. Now pick a random l-element subset I of the set $[1, n]$, and, for each $i \in I$, a random element $\nu_i \xleftarrow{R} B$. Let Q be the set $\{(i, \nu_i)\}$. Send Q to the prover.

Parse the prover's response to obtain $(\mu_1, \ldots, \mu_s) \in (\mathbb{Z}_p)^s$ and $\sigma \in G$. If parsing fails, fail by emitting 0 and halting. Otherwise, check whether

$$e(\sigma, g) \stackrel{?}{=} e\Big(\prod_{(i,\nu_i) \in Q} H(name \| i)^{\nu_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j},\ v \Big) \ ;$$

if so, output 1; otherwise, output 0.

Pub.$\mathcal{P}(pk, t, M^*)$**.** Parse the processed file M^* as $\{m_{ij}\}$, $1 \le i \le n$, $1 \le j \le s$, along with $\{\sigma_i\}$, $1 \le i \le n$. Parse the message sent by the verifier as Q, an l-element set $\{(i, \nu_i)\}$, with the i's distinct, each $i \in [1, n]$, and each $\nu_i \in B$. Compute

$$\mu_j \leftarrow \sum_{(i,\nu_i) \in Q} \nu_i m_{ij} \in \mathbb{Z}_p \quad \text{for } 1 \le j \le s, \qquad \text{and} \qquad \sigma \leftarrow \prod_{(i,\nu_i) \in Q} \sigma_i^{\nu_i} \in G \ .$$

Send to the prover in response the values $\mu_1, \ldots, \mu_s$ and σ.

[21] For notational simplicity, we present our scheme using a symmetric bilinear map, but efficient implementations will use an asymmetric map $e\colon G_1 \times G_2 \to G_T$. Translating our scheme to this setting is simple. User public keys v will live in G_2; file generators u_j will live in G_1, as will the output of H; and security will be reduced to co-CDH [5].

4 Security Proofs

In this section we prove that both of our systems are secure under the model we provided. Intutively, we break our proof into three parts. The first part shows that the attacker can never give a forged response back to the a verifier. The second part of the proof shows that from any adversary that passes the check a non-negligible amount of the time we will be able to extract a constant fraction of the encoded blocks. The second step uses the fact that (w.h.p.) all verified responses must be legitimate. Finally, we show that if this constant fraction of blocks is recovered we can use the erasure code to reconstruct the original file.

In this section we provide an outline of our proofs and state our main theorems and lemmas. We defer the proofs of these to the full paper [16]. The proof, for both schemes, is in three parts:

1. Prove that the verification algorithm will reject except when the prover's $\{\mu_j\}$ are correctly computed, i.e., are such that $\mu_j = \sum_{(i,\nu_i)\in Q} \nu_i m_{ij}$. This part of the proof uses cryptographic techniques.
2. Prove that the extraction procedure can efficiently reconstruct a ρ fraction of the file blocks when interacting with a prover that provides correctly-computed $\{\mu_j\}$ responses for a nonnegligible fraction of the query space. This part of the proof uses combinatorial techniques.
3. Prove that a ρ fraction of the blocks of the erasure-coded file suffice for reconstructing the original file. This part of the proof uses coding theory techniques.

Crucially, only the part-one proof is different for our two schemes; the other parts are *identical*.

4.1 Part-One Proofs

Scheme with Private Verifiability

Theorem 1. *If the MAC is unforgeable, the symmetric encryption scheme is semantically secure, and the PRF is secure, then (except with negligible probability) no adversary against the soundness of our private-verification scheme ever causes $\mathcal{V}$ to accept in a proof-of-retrievability protocol instance, except by responding with values $\{\mu_j\}$ and σ that are computed correctly, i.e., as they would be by $\mathsf{Priv}.\mathcal{P}$.*

We prove the theorem in Appendix A.1 of the full paper [16].

Scheme with Public Verifiability

Theorem 2. *If the signature scheme used for file tags is existentially unforgeable and the computational Diffie-Hellman problem is hard in bilinear groups, then, in the random oracle model, except with negligible probability no adversary against the soundness of our public-verification scheme ever ever causes $\mathcal{V}$ to accept in a proof-of-retrievability protocol instance, except by responding with values $\{\mu_j\}$ and σ that are computed correctly, i.e., as they would be by $\mathsf{Pub}.\mathcal{P}$.*

We prove the theorem in Appendix A.2 of the full paper [16].

4.2 Part-Two Proof

We say that a cheating prover $\mathcal{P}'$ is *well-behaved* if it never causes $\mathcal{V}$ to accept in a proof-of-retrievability protocol instance except by responding with values $\{\mu_j\}$ and σ that are computed correctly, i.e., as they would be by $\mathsf{Pub}.\mathcal{P}$. The part-one proofs above guarantee that all adversaries that win the soundness game with nonnegligible probability output cheating provers that are well-behaved, provided that the cryptographic primitives we employ are secure. The part-two theorem shows that extraction always succeeds against a well-behaved cheating prover:

Theorem 3. *Suppose a cheating prover $\mathcal{P}'$ on an n-block file M is well-behaved in the sense above, and that it is ϵ-admissible: i.e., convincingly answers and ϵ fraction of verification queries. Let $\omega = 1/\#B + (\rho n)^l/(n-l+1)^l$. Then, provided that $\epsilon - \omega$ is positive and nonnegligible, it is possible to recover a ρ fraction of the encoded file blocks in $O\big(n \;/\; (\epsilon - \omega)\big)$ interactions with $\mathcal{P}'$ and in $O\big(n^2 s + (1 + \epsilon n^2)(n) \;/\; (\epsilon - \omega)\big)$ time overall.*

We first make the following definition.

Definition 2. *Consider an adversary $\mathcal{B}$, implemented as a probabilistic polynomial-time Turing machine, that, given a query Q on its input tape, outputs either the correct response ($\boldsymbol{q}M$ in vector notation) or a special symbol $\perp$ to its output tape. Suppose $\mathcal{B}$ responds with probability ϵ, i.e., on an ϵ fraction of the query-and-randomness-tape space. We say that such an adversary is ϵ-polite.*

The proof of our theorem depends upon the following lemma that is proved in Appendix A.3 of the full paper [16].

Lemma 1. *Suppose that $\mathcal{B}$ is an ϵ-polite adversary as defined above. Let ω equal $1/\#B + (\rho n)^l/(n-l+1)^l$. If $\epsilon > \omega$ then it is possible to recover a ρ fraction of the encoded file blocks in $O\big(n \;/\; (\epsilon - \omega)\big)$ interactions with $\mathcal{B}$ and in $O\big(n^2 s + (1 + \epsilon n^2)(n) \;/\; (\epsilon - \omega)\big)$ time overall.*

To apply Lemma 1, we need only show that a well-behaved ϵ-admissible cheating prover $\mathcal{P}'$, as output by a setup-game adversary $\mathcal{A}$, can be turned into an ϵ-polite adversary $\mathcal{B}$. But this is quite simple. Here is how $\mathcal{B}$ is implemented. We will use the $\mathcal{P}'$ to construct the ϵ-adversary $\mathcal{B}$. Given a query Q, interact with $\mathcal{P}'$ according to $\big(\mathcal{V}(pk, sk, t, sk) \rightleftharpoons \mathcal{P}'\big)$, playing the part of the verifier. If the output of the interaction is 1, write $(\mu_1, \ldots, \mu_s)$ to the output tape; otherwise, write $\perp$. Each time $\mathcal{B}$ runs $\mathcal{P}'$, it provides it with a clean scratch tape and a new randomness tape, effectively rewinding it. Since $\mathcal{P}'$ is well-behaved, a successful response will compute $(\mu_1, \ldots, \mu_s)$ as prescribed for an honest prover. Since $\mathcal{P}'$ is ϵ-admissible, on an ϵ fraction of interactions it answers correctly. Thus algorithm $\mathcal{B}$ that we have constructed is an ϵ-polite advesrary.

All that remains to to guarantee that $\omega = 1/\#B + (\rho n)^l/(n-l+1)^l$ is such that $\epsilon - \omega$ is positive – indeed, nonnegligible. But this simply requires that each of $1/\#B$ and $(\rho n)^l/(n-l+1)^l$ be negligible in the security parameter; see Section 1.1.

4.3 Part-Three Proof

Theorem 4. *Given a ρ fraction of the n blocks of an encoded file M^*, it is possible to recover the entire original file M with all but negligible probability.*

Proof. For rate-ρ Reed-Solomon codes this is trivially true, since any ρ fraction of encoded file blocks suffices for decoding; see Appendix B of the full paper [16]. For rate-ρ linear-time codes the additional measures described there guarantee that the ρ fraction of blocks retrieved will allow decoding with overwhelming probability.

Acknowledgements

We thank Dan Boneh, Moni Naor, and Guy Rothblum for helpful discussions regarding this work, and Eric Rescorla for detailed comments on the manuscript.

References

1. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In S. De Capitani di Vimercati and P. Syverson, editors, *Proceedings of CCS 2007*, pages 598–609. ACM Press, Oct. 2007.
2. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. Cryptology ePrint Archive, Report 2007/202, 2007. Online: `http://eprint.iacr.org/`. Version of 7 Dec. 2007; visited 10 Feb. 2008.
3. G. Ateniese, R. Di Pietro, L. Mancini, and G. Tsudik. Scalable and efficient provable data possession. In P. Liu and R. Molva, editors, *Proceedings of SecureComm 2008*. ICST, Sept. 2008. To appear.
4. P. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Proceedings of SAC 2005*, volume 3897 of *LNCS*, pages 319–31. Springer-Verlag, 2006.
5. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, Sept. 2004. Extended abstract in *Proceedings of Asiacrypt 2001*.
6. K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: Theory and implementation. Cryptology ePrint Archive, Report 2008/175, 2008. `http://eprint.iacr.org/`.
7. R. Cramer, G. Hanaoka, D. Hofheinz, H. Imai, E. Kiltz, R. Pass, abhi shelat, and V. Vaikuntanathan. Bounded CCA2-secure encryption. In K. Kurosawa, editor, *Proceedings of Asiacrypt 2007*, volume 4833 of *LNCS*, pages 502–18. Springer-Verlag, Dec. 2007.
8. Y. Deswarte, J.-J. Quisquater, and A. Saïdane. Remote integrity checking. In S. Jajodia and L. Strous, editors, *Proceedings of IICIS 2003*, volume 140 of *IFIP*, pages 1–11. Kluwer Academic, Jan. 2004.
9. D. Filho and P. Barreto. Demonstrating data possession and uncheatable data transfer. Cryptology ePrint Archive, Report 2006/150, 2006. `http://eprint.iacr.org/`.

10. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. Cryptology ePrint Archive, Report 2006/372, 2006. http://eprint.iacr.org/.
11. S.-H. Heng and K. Kurosawa. k-resilient identity-based encryption in the standard model. *IEICE Trans. Fundamentals*, E89-A.1(1):39–46, Jan. 2006. Originally published at CT-RSA 2004.
12. A. Juels and B. Kaliski. PORs: Proofs of retrievability for large files. In S. De Capitani di Vimercati and P. Syverson, editors, *Proceedings of CCS 2007*, pages 584–97. ACM Press, Oct. 2007. Full version: http://www.rsa.com/rsalabs/staff/bios/ajuels/publications/pdfs/POR-preprint-August07.pdf.
13. M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard. A cooperative Internet backup scheme. In B. Noble, editor, *Proceedings of USENIX Technical 2003*, pages 29–41. USENIX, June 2003.
14. M. Naor and G. Rothblum. The complexity of online memory checking. In E. Tardos, editor, *Proceedings of FOCS 2005*, pages 573–84. IEEE Computer Society, Oct. 2005.
15. T. Schwarz and E. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In M. Ahamad and L. Rodrigues, editors, *Proceedings of ICDCS 2006*. IEEE Computer Society, July 2006.
16. H. Shacham and B. Waters. Compact proofs of retrievability. Cryptology ePrint Archive, Report 2008/073, 2008. http://eprint.iacr.org/.
17. M. Shah, R. Swaminathan, and M. Baker. Privacy-preserving audit and extraction of digital contents. Cryptology ePrint Archive, Report 2008/186, 2008. http://eprint.iacr.org/.