Compact Proofs of Retrievability

Hovav Shacham
hovav@cs.ucsd.edu

Brent Waters
bwaters@csl.sri.com

February 20, 2008

Abstract

In a proof-of-retrievability system, a data storage center must prove to 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. All previous provably secure solutions require that a prover send $O(l)$ authenticator values (i.e., MACs or signatures) to verify a file, for a total of $O(l^2)$ bits of communication, where l is the security parameter. The extra cost over the ideal $O(l)$ communication can be prohibitive in systems where a verifier needs to check many files.

We create the first compact and provably secure proof of retrievability systems. Our solutions allow for compact proofs with just one authenticator value — in practice this can lead to proofs with as little as 40 bytes of communication. We present two solutions with similar structure. The first one is privately verifiable and builds elegantly on pseudorandom functions (PRFs); the second allows for publicly verifiable proofs and is built from the signature scheme of Boneh, Lynn, and Shacham in bilinear groups. Both solutions rely on homomorphic properties to aggregate a proof into one small authenticator value.

1 Introduction

There is a increasing trend of both personal and enterprise data's being stored at third party locations. In some cases users employ explicit storage services such as Amazon's S3 online storage service to back up data. In other cases, user data will be implicitly stored at third party sites as part of the trend of "Software as a Service." For example, the site `Salesforce.com` provides a suite of online tools that helps to manage several aspects of a corporation's sales strategy and stores all stores all sales records submitted.

In several applications it is essential for a user to be ensure himself that his data is still available and ready for retrieval if needed. E.g., he wants to make sure that the service didn't lose his e-mail or sales invoices. This 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 being saved.

Recently, Juels and Kaliski [17] and independently Ateniese et al. [4, 3] looked at how the problem of how a server can prove that it is storing data. Juels and Kaliski cast this problem as a "Proof of Retrievability" (POR) and provided a formal definition for a secure POR system — in a secure system if a server can pass an audit then a simulator (w.h.p.) must be able to extract the file.

One simple way to achieve this is to demand that the service send back all the data it claims to have stored for the user[1]; however this auditing method can be prohibitively expensive in practice. Many times, a user will want only to audit the server, not to retrieve all of her data — of which there might be terabytes. Ideally, a proof-of-retrievability system will satisfy the following desiderata:

Efficient. The system should be as efficient as possible in terms of both computational complexity and communication complexity. Ideally, a system should have $O(l)$ bits of communication for a POR of a file, where l is a security parameter.

Publicly verifiable. A system is publicly verifiable if any (untrusted) entity can perform the verification audit. This is desirable in settings where many users might share file storage or when a third party is employed to audit the storage servers.

Publicly retrievable. A system is publicly retrievable if an (untrusted) entity could potentially extract the file contents. In practice we could of course add additional access controls to prevent unauthorized access. A desirable property in this context is that if an attacker were allowed to access the stored file he still could not break the POR security.

Unbounded use. We would like a system not to impose an *a priori* bound on the number of audit protocol interactions. In the past bounded systems have been considered for problems such as Identity-Based Encryption [16] and CCA-secure encryption [9] where the system is secure as long as the adversary makes at most t private key extraction or decryption queries (typically the public parameters in such systems will grow with t). After t queries the system will need to stop functioning; accordingly, unbounded use has traditionally been considered to be the "right" definition for such systems and we consider it to be important in POR systems.

Stateless. A stateless verifier will not need to maintain state between different audits. Publicly verifiable systems can be inherently stateless. In private verification systems a stateless system can be audited by multiple trusted machines without requiring coordination.

In their paper, Juels and Kaliski provide two schemes in which they redundantly encode a file with an erasure code and apply an audit that probabilistically ensures enough blocks are retrievable to reconstruct the file. In their first scheme, the encoder embeds "sentinel" blocks into the file before encrypting it. These blocks are derived from a PRF for which the verifier keeps the key. During an audit the verifier requests a set of sentinel blocks that haven't been used before and checks them for correctness. Since the file is encrypted the server cannot distinguish sentinel values from data blocks and a server that corrupts or forgets enough data blocks with high probability will corrupt a sentinel block and get caught.

The primary drawback of this first solution is that some sentinel blocks get used up with each audit. Thus, a verifier can perform only a limited number of audits and the system only meets a weaker, bounded correctness definition; moreover, the verifier must hold a secret and be stateful. Therefore, the verifying party must be trusted and even multiple trusted machines cannot independently verify as they must share state (or pre-divide the set of sentinels used). To address these concerns Juels and Kaliski propose a second scheme, using techniques from Lillibridge et al. [19] and Naor and Rothblum [22], in which each block of the redundantly encoded data is stored along with a

[1]The user can easily verify the integrity of the response if he stores a hash of the file. Alternatively, the user can store a signature alongside the data at the server and verify the signature upon retrieval.

MAC. (Signatures can also be used to obtain public verifiability; Ateniese et al. [4, 3] first suggested publicly verifiable schemes.) In an audit, the verification algorithm requests a random l-element set of blocks from the receiver, together with their MACs, and checks the integrity of each block.

While this second solution addresses the problems enumerated above, the communication complexity of audits is relatively large. In particular, in order to achieve l-bit security the prover must send the verifier $O(l)$ blocks. In practice, this can be a problem for applications in which a user wants to audit several different stored files — each file will require around l MACs or approximately $O(l^2)$ bits of communication. For typical security parameters this can blow up the communication by a factor of around 80 compared to an ideal solution where only one MAC is sent.

1.1 Our Contributions

In this paper we create the first systems with compact proofs of retrievability. We present two solutions with similar structure.

A scheme with private verifiability. Our first solution is privately verifiable and can admit proofs of retrievability as small as 20 bytes. Our main idea is that instead of sending out l separate authentication-value–message-block pairs (for security parameter l), the prover aggregates them into one pair using homomorphic operations, then sends out the aggregate, which is only as long as a single block and MAC.

We realize this by breaking an (erasure encoded) file into n blocks $m_1, \ldots, m_n \in Z_p$ for some large prime p. The storage service will then choose a random $\alpha \in \mathbb{Z}_p$ and PRF key k for function f. These are the secret keys for the system. Each block will then be associated with an authentication value $\sigma_i \in \mathbb{Z}_p$, where σ_i is calculated as

$$\sigma_i = f_k(i) + \alpha m_i \ .$$

When the verification algorithm requests a proof it will specify a random challenge set I of l indices along with l random coefficients in Z_p. In the random oracle model we can use a hash function to select all of these challenge values with small communication complexity. Let Q be the set $\{(i, \nu_i)\}$ of challenge index–coefficient pairs. The prover then calculates the response, a pair (σ, μ), as

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

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

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

While it is clear that our techniques admit short responses, it is not obvious that our new system will admit a simulator that can extract files. If the prover returned l separate blocks in each verification response then an extractor could interact with the prover several times and collect enough blocks to reconstruct the file. (By making l appropriately large the verifier can ensure, w.h.p., that the prover has enough of the encoded blocks.) However, when the blocks are aggregated together, as in our case, a proof of extraction becomes more difficult. It isn't immediately obvious that any compression strategy will work at all — it may possible for an adversary to store many

different linear combinations of the blocks and use these to answer queries, but not know any individual data blocks. Indeed, in Appendix C we show such an attack on a system in which the challenge coefficient values aren't chosen correctly.

To prove security, there are two new challenges that we must overcome. First, we must ensure that any response (σ, μ) given to a challenge is correct. To do this we argue that if the PRF is indistinguishable from a truly random function then an adversary will have no knowledge of α and has at most a $1/p$ chance of forgery on any response. The second part of the argument is more challenging. In our proof we view use a simulator that attempts to extract the file by repeatedly recovering different linear combinations of blocks. These can be viewed as vectors over the space of the encoded file. Gradually, the simulator will gather many of these vectors. In our proof we need to make combinatorial arguments to show that the simulator can retrieve enough *individual* blocks to reconstruct the file. The proof does not make use of random oracles; for details, see Section 4.

A scheme with public verifiability. Our second scheme is publicly verifiable. It follows the same framework as the first, but instead uses BLS signatures [6] 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 problem over bilinear groups in the random oracle model. In Appendix D we present a variant of the second scheme that is less efficient but is secure under the RSA assumption.

Our proof. We provide a modular proof for the security of our schemes, with three parts based respectively on cryptographic, combinatorial, and coding-theoretical techniques. Only the first part differs between the three schemes we propose. Our analysis is more precise than in previous work; for example, we show that, for 80-bit security, challenge coefficients ν_i can be 80 bits long, not 160 as proposed in [3, p. 17]. Moreover, our analysis shows how system parameters can be reduced if applications are willing to accept a 1-in-1,000,000 chance, say that an adversary will manage to answer a proof-of-storage challenge correctly although he is not storing the file.

1.2 Related Work

There have been systems proposed with the aim of proving that a file is stored remotely [12, 13, 25]. While the goals of these papers are akin to proofs of retrievability, none of them gives a formal proof that stored data could be extracted from a server that responds to audits.

As stated above, Juels and Kaliski [17] gave the first formal definitions of extractability. In their work they presented two systems; the first only achieved a weaker bounded notion of correctness (and wasn't publicly verifiable or retrievable), while the second (using ideas from Lillibridge et al. [19] and Naor and Rothblum [22]) can be made unbounded with public verifiability and retrievability. However, in this scheme the prover needs to send $O(l)$ authenticator values and thus isn't compact. It is interesting to note that in the first sentinel-based scheme Juels and Kaliski suggest a variant in which authenticator values are randomly XORed together to give a compact proof. The authors do not, however, give a proof that in this variant the file can be extracted. As we will see, proving extraction from aggregated authenticator values can be challenging; in Appendix C we show an attack on a natural but incorrect system. If the the XOR variant turned out to be secure, it would still have the limitation of working for only a bounded number of queries.

Other work [12, 13] has considered RSA-based authenticators that exponentiate over the *entire* file. Recently, Ateniese et al. [4, 3] proposed an interesting system that applied an RSA authenti-

cator. They leverage the homomorphic properties of the RSA authenticator to compress the server response. Roughly, In their system a file is parsed into n blocks of $\mathbb{Z}_N$ with an authenticator attached to each block. A prover proves possession of a file by homomorphically combining the authenticators of each block. They proved storage possession under the RSA assumption.

The primary drawback of this first approach is its computational demands on the prover: an exponentiation for each block of data. For example, if we assume that a full exponentiation takes 5 ms for a 1024-bit RSA modulus, then proving possession of a 1 GB file will take approximately 11 hours.

To deal with this issue Ateniese et al. [4, 3] suggest that one might modify their system to sample some size-c subset of the blocks, where c is less than n. In addition, the authors give an implementation and measurements of their system with relatively small values of c. For smaller values of c one must provide erasure encoding for an audit to ensure that a file is actually stored. For example, in the absence of encoding, if any one block is erased by the storage server then this block will be lost. If an adversary erases one block he will have a $\frac{n-c}{n}$ chance of escaping an audit undetected, since this is the probability that the block will not be included in the audit.

While the authors suggest that erasure encoding might be used in such a system, they do not provide a proof that in such a system with erasure encoding will necessarily be secure against *all* attackers — that is, prove that for *any* attacker a simulator can extract the file. The authors give, in Section 5, an analysis for an adversary that "deletes" certain blocks and keeps others. This doesn't model all the way an adversary might choose to misbehave. For example, an adversary might store certain linear combinations of blocks (e.g., $4 \cdot m_1 + 3 \cdot m_3$) that he could use to answer queries. In this ambiguous case the attacker neither has blocks m_1 and m_3 since they are combined together, nor has he completely deleted either one of them since the information he has about them can be used to answer some queries. In Appendix C we describe a natural scheme that falls to just such an attack. Moreover, it is not completely clear how to define an adversary that "deletes" certain blocks and maintains others. For example, suppose some block comes from a low entropy distribution (e.g., there are only two possible values). An adversary might be able to delete that block — and lose information — and nevertheless might pass a particular audit simply by guessing what the missing block was. (This example also illustrates the importance of handling adversaries that succeed a fraction of the time, as an attacker might that makes such guesses.)

In general an attacker can use many different strategies (including economically beneficial ones) to attack a system. To say that a system is secure it is necessary to prove it secure against all such attackers. We consider providing compact and efficient systems with proofs of security against *any* adversary our main contribution.

We emphasize that these arguments do not necessarily mean that there exist attacks on the Ateniese et al. system (when erasure codes are applied); however, in order to prove such schemes secure one must consider all such attackers. Indeed, in addition to our main schemes, we also show, in Appendix D, how full-domain–hash RSA signatures [5, 8] can be used in our framework as an alternative to BLS signatures. Our RSA-based variant is structurally similar to the Ateniese et al. scheme and can be viewed as evidence that their scheme would be secure when instantiated with small c values, if erasure encoding were correctly applied.

2 Security Model

A proof of retrievability scheme defines four algorithms: Kg, St, $\mathcal{V}$, and $\mathcal{P}$. These behave as follows.

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: $\{0,1\} \xleftarrow{\mathrm{R}} \big(\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{P}(pk, t, M^*)\big)$.

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_i. 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 $\mathcal{P}'$.

Let M be the message input to the store query that returned the challenge tag t (along with a processed version M^* of M).

The cheating prover $\mathcal{P}'$ is ϵ-*admissible* if it convincingly answers an ϵ fraction of verification challenges, i.e., if $\Pr\left[\left(\mathcal{V}(pk,t) \rightleftharpoons \mathcal{P}'\right) = 1\right] \geq \epsilon$. Here the probability is over the coins of the verifier and the prover.

Definition 2.1. 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., $\mathsf{Extr}(pk, t, \mathcal{P}') = M$ — 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.

2.1 Notes on the Model

Differences from Juels-Kaliski model. We briefly compare our definition to that given in [17]. Like the Juels-Kaliski definition, our definition is intended to capture the intuition that extractability is the requirement for a proof of retrievability. Unlike Juels and Kaliski, 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.

In addition, we do not provide any of the algorithms with mutable state. Verifiers are stateless (and probabilistic) in our schemes, and, as we noted in Section 1, we believe that statelessness is important, particularly when there are multiple verifiers.

Finally, we require that extraction succeed (with all but negligible probability) from an adversary that causes $\mathcal{V}$ to accept with any nonnegligible probability ϵ. 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. At the same time, we do not make any block-isolation or independence assumptions about the adversary's behavior; extraction requires only that the $\left(\mathcal{V}(pk,t) \rightleftharpoons \mathcal{P}'\right)$ interaction output 1 on an ϵ fraction of the query-and-randomness-tape space.

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^k for security parameter k, we require each algorithm to run in time polynomial in k, and we require that extraction fail from an ϵ-admissible prover with only negligible probability in k, 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 blocks as $\{m_{ij}\}$, with $1 \leq i \leq n$ and $1 \leq j \leq s$.

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 $\mathbf{q} \in (\mathbb{Z}_p)^n$ where $\mathbf{q}_i = \nu_i$ for $i \in I$ and $\mathbf{q}_i = 0$ for all $i \notin I$. Equivalently, letting $\mathbf{u}_1, \ldots, \mathbf{u}_n$ be the usual basis for $(\mathbb{Z}_p)^n$, we have $\mathbf{q} = \sum_{(i,\nu_i) \in Q} \nu_i \mathbf{u}_i$.[2]

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 "$i \notin I$."

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

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 $\mathbf{q}M$.

3.2 Construction for Private Verification

Let $f\colon \{0,1\}^* \times \mathcal{K}_{\text{prf}} \to \mathbb{Z}_p$ be a PRF.[3] 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.

[3] 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. 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: $t \leftarrow t_0 \| \mathsf{SSig}_{ssk}(t_0)$. Now, 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.

3.4 Parameter Choices

Let λ be a security parameter; typically, $\lambda = 80$. For the private verification system, p must be λ bits for the MAC to be secure.[4] For the public verification system, p must be large enough that discrete logarithm is hard in the bilinear group G: 2λ bits suffice for typical values of λ [14].

[4] In fact, any λ-bit field will do; $\mathrm{GF}(2^\lambda)$ would be particularly suitable for hardware implementation.

The choice in s gives a tradeoff between storage overhead and computational and communication overhead. For most practical systems we expect s to be small.

The challenge set B, the coding rate ρ, and the size of the challenge set l are chosen depending on ϵ, the maximum acceptable false-accept rate, per Theorem 4.3. Assuming $n \gg l$, we need $1/\#B + \rho^l \ll \epsilon$. Since ϵ is nonnegligible, this will certainly be satisfied when B is the set of λ-bit strings, $\rho = 1/2$, and l equals λ. For applications that can tolerate a larger error rate these parameters can be reduce. For example, if a 1-in-1,000,000 error is acceptable, we can take B to be the set of 12-bit strings, and l to be 12; alternatively, the coding expansion $1/\rho$ can be reduced. Note that the smaller the elements of B, the more efficient the multiplications or exponentiations that involve them.

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 Appendix A. 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 non-negligible 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.

The crucial point is the the second and third parts of the proof are *identical* for our two schemes; only the first part is different.

4.1 Part-One Proofs

4.1.1 Scheme with Private Verifiability

Theorem 4.1. *If the MAC scheme 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.

4.1.2 Scheme with Public Verifiability

Theorem 4.2. *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.

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 4.3. *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 $\mathcal{A}$ and in $O\big(n^2 s + (1+\epsilon n^2)(n) \,/\, (\epsilon-\omega)\big)$ time overall.*

We first make the following definition.

Definition 4.4. 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 ($\mathbf{q}M$ 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

Lemma 4.5. *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 4.5, 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 3.4. □

4.3 Part-Three Proof

Theorem 4.6. *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. For rate-ρ linear-time codes the additional measures described in Appendix B.1 guarantee that the ρ fraction of blocks retrieved will allow decoding with overwhelming probability. □

References

[1] M. Aigner and G. Ziegler. *Proofs from The Book*. Springer-Verlag, 3rd edition, 2004.

[2] N. Alon and M. Luby. A linear time erasure-resilient code with nearly optimal recovery. *IEEE Trans. Info. Theory*, 42(6):1732–6, Nov. 1996.

[3] 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.

[4] 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.

[5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. Denning, R. Pyle, R. Ganesan, R. Sandhu, and V. Ashby, editors, *Proceedings of CCS 1993*, pages 62–73. ACM Press, Nov. 1993.

[6] 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*.

[7] H. Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, 1993.

[8] J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 229–35. Springer-Verlag, Aug. 2000.

[9] 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.

[10] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Trans. Info. & System Security*, 3(3):161–85, 2000.

[11] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Computing*, 33(1):167–226, 2003.

[12] 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.

[13] D. Filho and P. Barreto. Demonstrating data possession and uncheatable data transfer. Cryptology ePrint Archive, Report 2006/150, 2006. `http://eprint.iacr.org/`.

[14] 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/`.

[15] L. Guillou and J.-J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In C. Günther, editor, *Proceedings of Eurocrypt 1988*, volume 330 of *LNCS*, pages 123–8. Springer-Verlag, May 1988.

[16] 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.

[17] 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 584–597. ACM Press, Oct. 2007. Full version: `http://www.rsa.com/rsalabs/staff/bios/ajuels/publications/pdfs/POR-preprint-August07.pdf`.

[18] M. Krohn, M. Freedman, and D. Mazières. On-the-fly verification of rateless erasure codes for efficient content distribution. In D. Wagner and M. Waidner, editors, *Proceedings of IEEE Security & Privacy 2004*, pages 226–40. IEEE Computer Society, May 2004.

[19] 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.

[20] M. Liskov, R. Rivest, and D. Wagner. Tweakable block ciphers. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 31–46. Springer-Verlag, Aug. 2002.

[21] P. Maymounkov. Online codes. Technical Report TR2002-833, NYU, 2002.

[22] 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.

[23] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–48, Apr. 1989.

[24] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM SIGCOMM Computer Communication Rev.*, 27(2):24–36, Apr. 1997.

[25] 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.

A Security Proof Details

In this section we give the proofs of the theorems and lemmas stated in Section 4.

A.1 Proof of Theorem 4.1

Theorem 4.1. *If the MAC scheme 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 a series of games.

Game 0. The first game, Game 0, is simply the challenge game defined in Section 2.

Game 1. Game 1 is the same as Game 0, with one difference. The challenger keeps a list of all MAC-authenticated tags ever issued as part of a store-protocol query. If the adversary ever submits a tag t either in initiating a proof-of-storage protocol or as the challenge tag, that (1) verifies as valid under k_{mac} but (2) is not a tag authenticated by the challenger, the challenger declares failure and aborts.

Clearly, if there is a difference in the adversary's success probability between Games 0 and 1, we can use the adversary to construct a forger against the MAC scheme.

Game 2. In Game 2, the challenger includes in the tags not the encryption of $k_{\text{prf}}\|\alpha_1\|\cdots\|\alpha_s$ but a random bit-string of the same length. When given a tag by the adversary, the challenger uses the values that would (in previous games) have been encrypted in the tag.

Note that the modification made in Game 1 ensures that the challenger never sees a tag except those it issued through St queries, so it need never actually decrypt a tag portion.

Clearly, if there is a difference in the adversary's success probability between Games 0 and 1, we can use the adversary to break the semantic security of the symmetric encryption scheme. Note that the reduction so obtained will suffer a $1/q_S$ security loss, where q_S is the number of St queries made by the adversary, because we must use a hybrid argument between "all valid encryptions" and "no valid encryptions."

Game 3. In Game 3, the challenger picks uses truly random values in $\mathbb{Z}_p$ instead of PRF output, remembering these values to use when verifying the adversary's responses in proof-of-storage protocol instances. More specifically, the challenger evaluates $f_{k_{\text{prf}}}(i)$ not by applying the PRF algorithm but by generating a random value $r \xleftarrow{\text{R}} \mathbb{Z}_p$ and inserting an entry (k_{prf}, i, r) in a table; it consults this table when evaluating the PRF to ensure consistency.

If there is a difference in the adversary's success probability between Games 0 and 1, we can use the adversary to break the security of the PRF. As before, a hybrid argument necessitates a security loss in the reduction; this time, the loss is $1/(Nq_S)$, where N is a bound on the number of blocks in the encoding of any file the adversary requests to have stored.

Game 4. In Game 4, the challenger keeps a table of its responses to St queries made by the adversary. It observes each instance of the proof-of-storage protocol with the adversary — whether because of a proof-of-storage query made by the adversary, or in the test made of $\mathcal{P}'$, or as part of the extraction attempt by Extr. If in any of these interactions the adversary responds with in a way that (1) passes the verification algorithm but (2) is not what would have been computed by an honest prover, the challenger declares failure and aborts.

More specifically, suppose the protocol instance involves an n-block file with secret values $\alpha_1, \ldots, \alpha_s$, contains sectors $\{m_{ij}\}$, and the block signatures issued by St are $\{\sigma_i\}$. Suppose $Q = \{(i, \nu_i)\}$ is the query that causes the challenger to abort, and that the adversary's response to that query was $\mu'_1, \ldots, \mu'_s$ together with σ'. Let the expected response — i.e., the one that would have been obtained from an honest prover — be $\mu_1, \ldots, \mu_s$ and σ, where $\sigma = \sum_{(i,\nu_i)\in Q} \nu_i \sigma_i$ and $\mu_j = \sum_{(i,\nu_i)\in Q} \nu_i m_{ij}$ for $1 \le j \le s$. If the adversary's response satisfies the verifier — i.e., if $\sigma' = \sum_{(i,\nu_i)\in Q} \nu_i r_{k_{\mathrm{prf}},i} + \sum_{j=1}^{s} \alpha_j \mu'_j$, where $r_{k_{\mathrm{prf}},i}$ is the random value substituted by Game 2 for $f_{k_{\mathrm{prf}}}(i)$, but $\mu'_j \neq \mu_j$ for at least one j, the challenger aborts. (If $\mu'_j = \mu_j$ for all j but $\sigma' \neq \sigma$, it is impossible that the verification equation holds, so we need not worry about this case.)

By the correctness of the scheme the expected values σ along with $\{\mu_j\}$ also satisfy the verification equation, so we have $\sigma = \sum_{i\in I} r_{k_{\mathrm{prf}},i} + \sum_{j=1}^{s} \alpha_j \mu_j$. Letting $\Delta\sigma \stackrel{\mathrm{def}}{=} \sigma' - \sigma$ and $\Delta\mu_j \stackrel{\mathrm{def}}{=} \mu'_j - \mu_j$ for $1 \le j \le s$ and subtracting the verification equation for σ from that for σ', we have

$$\Delta\sigma = \sum_{j=1}^{s} \alpha_j \Delta\mu_j \ . \tag{1}$$

We can think of $\{\Delta\mu_j\}$ and $\Delta\sigma$ as variables in an $(s+1)$-dimensional space. The values for these variables that are accepted by the verifier are exactly those that satisfy equation (1), i.e., those that lie on a certain s-dimensional hyperplane. Any choice of values $\{\Delta\mu_j\}$ and $\Delta\sigma$ on this hyperplane, aside from the all-zero assignment that corresponds to the honest prover's answer, will cause the challenger to abort.

We will proceed by induction on the adversary's proof-of-storage protocol instances for any file, proving that if the adversary had not caused an abort in its first i protocol instances, it will not cause an abort in its $(i+1)$st protocol instance except with negligible probability.

At the time of the adversary's first protocol execution, the values $\alpha_1, \ldots, \alpha_s$ are independent of its view: They are no longer encrypted in the tag, and their only other appearance is in computing $\sigma_i = r_{k_{\mathrm{prf}},i} + \sum_{j=1}^{s} \alpha_j m_{ij}$ for $1 \le i \le n$; but the random value $r_{k_{\mathrm{prf}},i}$ replacing $f_{k_{\mathrm{prf}}}(i)$ (and used only there) means that σ_i is independent of $\alpha_1, \ldots, \alpha_j$. Thus the probability that the values $\{\Delta\mu_j\}$ and $\Delta\sigma$ it chooses lie on the abort hyperplane is $1/p$, so execution continues with probability $1 - 1/p$.

The adversary learns some information about the abort-hyperplane from the rejection of its response: he learns that the point he queried is not on the hyperplane, and neither is any point on the line between that point and the origin. For his second protocol execution, he can choose a point on a plane containing this line but not on the line; this point lies on the abort-hyperplane with probability $1/(p-1)$. No other way of choosing a point allows him a success probabilty so high.

More generally, after i executions, the adversary has eliminated i lines between his each of his i points and the origin. Thus for his $(i+1)$st execution he can succeed in naming a point on the abort-hyperplane with probability at most $1/(p-i)$. To achieve this bound, he must choose all his points from a single plane passing through the origin; with this strategy, each rejection allows him to eliminate p points out of the p^2 points on the plane.

To complete the induction, we see that the challenger aborts at some point during the adversary's q protocol executions with probability

$$1 - \left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{p-1}\right) \cdots \left(1 - \frac{1}{p-q+1}\right) \ ,$$

which is negligible. (This argument is inspired by Cramer and Shoup's analysis of their encryption scheme [11].)

Wrapping up. In Game 4, the adversary is constrained from answering any verification query with values other than those that would have been computed by $\mathsf{Priv}.\mathcal{P}$. Yet we have argued that, assuming the MAC, encryption scheme, and PRF are secure, there is only a negligible difference in the success probability of the adversary in this game compared to Game 1, where the adversary is not constrained in this manner. This completes the proof of Theorem 4.1. □

A.2 Proof of Theorem 4.2

Theorem 4.2. *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 $\mathsf{Pub}.\mathcal{P}$.*

Once again, we prove the theorem as a series of games.

Game 0. The first game, Game 0, is simply the challenge game defined in Section 2, with the changes for public verifiability sketched at the end of that section.

Game 1. Game 1 is the same as Game 0, with one difference. The challenger keeps a list of all signed tags ever issued as part of a store-protocol query. If the adversary ever submits a tag t either in initiating a proof-of-retrievability protocol or as the challenge tag, that (1) has a valid signature under *ssk* but (2) is not a tag signed by the challenger, the challenger declares failure and aborts.

Clearly, if there is a difference in the adversary's success probability between Games 0 and 1, we can use the adversary to construct a forger against the signature scheme.

Game 2. Game 2 is the same as Game 1, with one difference. The challenger keeps a list of its responses to St queries made by the adversary. Now the challenger observes each instance of the proof-of-retrievability protocol with the adversary — whether because of a proof-of-retrievability query made by the adversary, or in the test made of $\mathcal{P}'$, or as part of the extraction attempt by Extr. If in any of these instances the adversary is successful (i.e., $\mathcal{V}$ outputs 1) but the adversary's aggregate signature σ is not equal to $\prod_{(i,\nu_i)\in Q} \sigma_i^{\nu_i}$ (where Q is the challenge issued by the verifier and σ_i are the signatures on the blocks of the file considered in the protocol instance) the challenger declares failure and aborts.

Before analyzing the difference in success probabilities between Games 1 and 2, we will establish some notation and draw a few conclusions. Suppose the file that causes the abort is n blocks long, has name *name*, has generating exponents $\{u_j\}$, and contains sectors $\{m_{ij}\}$, and that the block signatures issued by St are $\{\sigma_i\}$. Suppose $Q = \{(i, \nu_i)\}$ is the query that causes the challenger

to abort, and that the adversary's response to that query was $\mu'_1, \ldots, \mu'_s$ together with σ'. Let the expected response — i.e., the one that would have been obtained from an honest prover — be $\mu_1, \ldots, \mu_s$ and σ, where $\sigma = \prod_{(i,\nu_i)\in Q} \sigma_i^{\nu_i}$ and $\mu_j = \sum_{(i,\nu_i)\in Q} \nu_i m_{ij}$ for $1 \le j \le s$. By the correctness of the scheme, we know that the expected response satisfies the verification equation, i.e., that

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

Because the challenger aborted, we know that $\sigma \neq \sigma'$ and that σ' passes the verification equation, i.e., that

$$e(\sigma', g) = e\Big(\prod_{(i,\nu_i)\in Q} H(name\|i)^{\nu_i} \cdot \prod_{j=1}^{s} u_j^{\mu'_j},\ v\Big) \ ,$$

where $v = g^\alpha$ is part of the challenger's public key. Observe that if $\mu'_j = \mu_j$ for each j, it follows from the verification equation that $\sigma' = \sigma$, which contradicts our assumption above. Therefore, if we define $\Delta\mu_j \stackrel{\text{def}}{=} \mu'_j - \mu_j$ for $1 \le j \le s$, it must be the case that at least one of $\{\Delta\mu_j\}$ is nonzero.

With this in mind, we now show that if there is a nonnegligible difference in the adversary's success probabilities between Games 1 and 2 we can construct a simulator that solves the computational Diffie-Hellman problem.

The simulator is given as inputs values $g, g^\alpha, h \in G$; its goal is to output h^α. The simulator behaves like the Game 1 challenger, with the following differences:

- In generating a key, it sets the public key v to g^α received in the challenge. This means that it does not know the corresponding secret key α.
- The simulator programs the random oracle H. It keeps a list of queries and responses to answers consistently. In answering the adversary's queries it chooses a random $r \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p$ and responds with $g^r \in G$. It also answers queries of the form $H(name\|i)$ in a special way, as we will see below.
- When asked to store some file whose coded representation comprises the n blocks $\{m_{ij}\}$, $1 \le i \le n$, $1 \le j \le s$, the simulator behaves as follows. It chooses a name *name* at random. Because the space from which names are drawn is large, it follows that, except with negligible probability, the simulator has not chosen this name before for some other file and a query has not been made to the random oracle at $name\|i$ for any i.

  For each j, $1 \le j \le s$, the simulator chooses random values $\beta_j, \gamma_j \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p$ and sets $u_j \leftarrow g^{\beta_j} \cdot h^{\gamma_j}$. For each i, $1 \le i \le n$, the simulator chooses a random value $r_i \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p$, and programs the random oracle at i as

$$H(name\|i) = g^{r_i} \,/\, \big(g^{\sum_{j=1}^s \beta_j m_{ij}} \cdot h^{\sum_{j=1}^s \gamma_j m_{ij}}\big) \ .$$

  Now the simulator can compute σ_i, since we have

$$\begin{aligned} H(name\|i) \cdot \prod_{j=1}^{s} u_j^{m_{ij}} &= \prod_{j=1}^{s} u_j^{m_{ij}} \cdot g^{r_i} \,/\, g^{\sum_{j=1}^s \beta_j m_{ij}} \cdot h^{\sum_{j=1}^s \gamma_j m_{ij}} \\ &= g^{\sum_{j=1}^s \beta_j m_{ij}} \cdot h^{\sum_{i=1}^s \gamma_j m_{ij}} \cdot g^{r_i} \,/\, g^{\sum_{j=1}^s \beta_j m_{ij}} \cdot h^{\sum_{j=1}^s \gamma_j m_{ij}} = g^{r_i} \ , \end{aligned}$$

  so the simulator computes $\sigma_i = \big(H(name\|i) \cdot \prod_{j=1}^s u_j^{m_{ij}}\big)^\alpha = (g^\alpha)^{r_i}$.

- The simulator continues interacting with the adversary until the condition specified in the definition of Game 2 occurs: the adversary, as part of a proof-of-retrievability protocol, succeeds in responding with a signature σ' that is different from the expected signature σ.

  The change made from Game 0 to Game 1 establishes that the parameters associated with this protocol instance — *name*, n, $\{u_j\}$, $\{m_{ij}\}$, and $\{\sigma_i\}$ — were generated by the simulator as part of a St query; otherwise, execution would have already aborted. This means that these parameters were generated according to the simulator's procedure described above. Now, dividing the verification equation for the forged signature σ' by the verification equation for the expected signature σ, we obtain

$$e(\sigma'/\sigma, g) = e\big(\prod_{j=1}^{s} u_j^{\Delta\mu_j}, v\big) = e\big(\prod_{j=1}^{s} (g^{\beta_j} \cdot h^{\gamma_j})^{\Delta\mu_j}, v\big) \ .$$

  Rearranging terms yields

$$e\big(\sigma' \cdot \sigma^{-1} \cdot v^{-\sum_{j=1}^{s} \beta_j \Delta\mu_j},\, g\big) = e(h, v)^{\sum_{j=1}^{s} \gamma_j \Delta\mu_j} \ ,$$

  Noting that v equals g^{α}, we see that we have found the solution to the computational Diffie-Hellman problem,

$$h^{\alpha} = \Big(\sigma' \cdot \sigma^{-1} \cdot v^{-\sum_{j=1}^{s} \beta_j \Delta\mu_j}\Big)^{\frac{1}{\sum_{j=1}^{s} \gamma_j \Delta\mu_j}}$$

  unless evaluating the exponent causes a divide-by-zero. However, we noted already that not all of $\{\Delta\mu_j\}$ can be zero, and the values of $\{\gamma_j\}$ are information theoretically hidden from the adversary,[5] so the denominator is zero only with probability $1/p$, which is negligible.

Thus if there is a nonnegligible difference between the adversary's probabilities of success in Games 1 and 2, we can construct a simulator that uses the adversary to solve computational Diffie-Hellman, as required.

Game 3. Game 3 is the same as Game 2, with one difference. As before, the challenger tracks St queries and observes proof-of-retrievability protocol instances. This time, if in any of these instances the adversary is successful (i.e., $\mathcal{V}$ outputs 1) but at least one of the aggregate messages m_j is not equal to the expected $\sum_{(i,\nu_i)\in Q} \nu_i m_{ij}$ (where, again, Q is the challenge issued by the verifier) the challenger declares failure and aborts.

Again, let us establish some notation. Suppose the file that causes the abort is n blocks long, has name *name*, has generating exponents $\{u_j\}$, and contains sectors $\{m_{ij}\}$, and that the block signatures issued by St are $\{\sigma_i\}$. Suppose $Q = \{(i, \nu_i)\}$ is the query that causes the challenger to abort, and that the adversary's response to that query was $\mu'_1, \ldots, \mu'_s$ together with σ'. Let the expected response — i.e., the one that would have been obtained from an honest prover — be $\mu_1, \ldots, \mu_s$ and σ, where $\sigma = \prod_{(i,\nu_i)\in Q} \sigma_i^{\nu_i}$ and $\mu_j = \sum_{(i,\nu_i)\in Q} \nu_i m_{ij}$ for $1 \le j \le s$. Game 2 already guarantees that we have $\sigma' = \sigma$; it is only the values $\{\mu'_j\}$ and $\{\mu_j\}$ that can differ. Define $\Delta\mu_j \stackrel{\text{def}}{=} \mu'_j - \mu_j$ for $1 \le j \le s$; again, at least one of $\{\Delta\mu_j\}$ is nonzero.

We now show that if there is a nonnegligible difference in the adversary's success probabilities between Games 2 and 3 we can construct a simulator that solves the discrete logarithm problem.

[5] Hidden because they are used to compute only the values $\{u_j\}$ in the adversary's view, and these are Pedersen commitments and so information-theoretically hiding.

The simulator is given as inputs values $g, h \in G$; its goal is to output x such that $h = g^x$. The simulator behaves like the Game 2 challenger, with the following differences:

- When asked to store some file whose coded representation comprises the n blocks $\{m_{ij}\}$, $1 \le i \le n$, $1 \le j \le s$, the simulator behaves according to St, except that For each j, $1 \le j \le s$, the simulator chooses random values $\beta_j, \gamma_j \xleftarrow{R} \mathbb{Z}_p$ and sets $u_j \leftarrow g^{\beta_j} \cdot h^{\gamma_j}$.

- The simulator continues interacting with the adversary until the condition specified in the definition of Game 3 occurs: the adversary, as part of a proof-of-retrievability protocol, succeeds in responding with aggregate messages $\{\mu'_j\}$ that are different from the expected aggregate messages $\{\mu_j\}$.

  As before, we know because of the change made in Game 1 that the parameters associated with this protocol instance were generated by the simulator as part of a St query. Because of the change made in Game 2 we know that $\sigma' = \sigma$. Equating the verification equations using $\{\mu'_j\}$ and $\{\mu_j\}$ gives us

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

  from which we conclude that

$$\prod_{j=1}^{s} u_j^{\mu_j} = \prod_{j=1}^{s} u_j^{\mu'_j}$$

  and therefore that

$$1 = \prod_{j=1}^{s} u_j^{\Delta\mu_j} = \prod_{j=1}^{s} \big(g^{\beta_j} \cdot h^{\gamma_j}\big)^{\Delta\mu_j} = g^{\sum_{j=1}^{s} \beta_j \Delta\mu_j} \cdot h^{\sum_{j=1}^{s} \gamma_j \Delta\mu_j}$$

  We see that we have found the solution to the discrete logarithm problem,

$$h = g^{-\frac{\sum_{j=1}^{s} \beta_j \Delta\mu_j}{\sum_{j=1}^{s} \gamma_j \Delta\mu_j}}$$

  unless the denominator is zero. However, not all of $\{\Delta\mu_j\}$ can be zero, and the values of $\{\gamma_j\}$ are information theoretically hidden from the adversary, so the denominator is zero only with probability $1/p$, which is negligible.

Thus if there is a nonnegligible difference between the adversary's probabilities of success in Games 2 and 3, we can construct a simulator that uses the adversary to compute discrete logarithms, as required.

Wrapping up. In Game 3, the adversary is constrained from answering any verification query with values other than those that would have been computed by $\mathsf{Pub}.\mathcal{P}$. Yet we have argued that, assuming the signature scheme is secure and computational Diffie-Hellman and discrete logarithm are hard in bilinear groups, there is only a negligible difference in the success probability of the adversary in this game compared to Game 1, where the adversary is not constrained in this manner. Moreover, the hardness of the CDH problem implies the hardness of the discrete logarithm problem. This completes the proof of Theorem 4.2. □

A.3 Proof of Lemma 4.5

To prove the lemma, we must first introduce some arguments in linear algebra.

For a subspace $\mathbb{D}$ of $(\mathbb{Z}_p)^n$, denote the dimension of $\mathbb{D}$ by $\dim \mathbb{D}$. Furthermore, let the free variables of a space, $\operatorname{free} \mathbb{D}$, be the indices of the basis vectors $\{\mathbf{u}_i\}$ included in $\mathbb{D}$, i.e.,

$$\operatorname{free} \mathbb{D} \stackrel{\text{def}}{=} \left\{ i \in [1,n] : \mathbb{D} \cap \mathbf{u}_i = \mathbf{u}_i \right\} \ .$$

Observe that if we represent $\mathbb{D}$ by means of a basis matrix in row-reduced echelon form, then we can efficiently compute $\dim \mathbb{D}$ and $\operatorname{free} \mathbb{D}$.

Next, we give two claims.

Claim A.1. *Let $\mathbb{D}$ be a subspace of $(\mathbb{Z}_p)^n$, and let I be an l-element subset of $[1,n]$. If $I \not\subseteq \operatorname{free} \mathbb{D}$, then a random query over indices I with coefficients in B is in $\mathbb{D}$ with probability at most $1/\#B$.*

Proof. Let $\mathbb{I}$ be the subspace spanned by the unit vectors in I, i.e., by $\{\mathbf{u}_i\}_{i \in I}$. Clearly, $\dim \mathbb{D} \cap \mathbb{I}$ is at most $l-1$; if it equalled l, then we would have $\mathbb{D} \cap \mathbb{I} = \mathbb{I}$ and each of the vectors $\{\mathbf{u}_i\}_{i \in I}$ would be in $\mathbb{D}$, contradicting the lemma statement. Suppose $\dim \mathbb{D} \cap \mathbb{I}$ equals r. Then there exist r indices in I such that a choice of values for the coordinates at these indices determines the values of the remaining $l-r$ coordinates. This means that there are at most $(\#B)^r$ vectors in $\mathbb{D} \cap \mathbb{I}$ with coordinate values in B: a choice of one of $\#B$ values for each of the r coordinates above determines the value to each of the other $l-r$ coordinates; if the values of these coordinates are all in B, then this vector contributes 1 to the count; otherwise it contributes 0. The maximum possible count is thus $(\#B)^r$. By contrast, there are $(\#B)^l$ vectors in $\mathbb{I}$ with coordinates in B, and these are exactly the vectors corresponding to each random query with indices I. Thus the probability that a random query is in $\mathbb{D}$ is at most $1 \,/\, (\#B)^{l-r} \leq 1 \,/\, (\#B)$, which proves the lemma. □

Claim A.2. *Let $\mathbb{D}$ be a subspace of $(\mathbb{Z}_p)^n$, and suppose that $\#(\operatorname{free} \mathbb{D}) = m$. Then for a random l-element subset I of $[1,n]$ the probability that $I \subseteq \operatorname{free} \mathbb{D}$ is at most $m^l/(n-l+1)^l$.*

Proof. Color the m indices included in $\operatorname{free} \mathbb{D}$ black; color the remaining $n-m$ indices white. A query I corresponds to a choice of l indices out of all these, without replacement. A query satisfies the condition that $I \subseteq \operatorname{free} \mathbb{D}$ exactly if every element of I is in $\operatorname{free} \mathbb{D}$, i.e., is colored black. Thus the probability that a random query satisfies the condition is just the probability of drawing l black balls, without replacement, from a jar containing m black balls and $n-m$ white balls; and this probability is

$$\binom{m}{l} \Big/ \binom{n}{l} = \frac{\big(m!/(m-l)!\big)}{\big(n!/(n-l)!\big)} < \frac{m^l}{(n-l+1)^l} \ ,$$

as required. Note that if $m > n-l$ then we will have $I \subseteq \operatorname{free} \mathbb{D}$ with probability 1; the upper bound given by this lemma is a probability greater than 1 in this case, but of course still correct. □

Lemma 4.5. *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.*

Proof. We say the extractor's knowledge at each point is a subspace $\mathbb{D}$, represented by a $t \times n$ matrix A in row-reduced echelon form. Suppose that the query–response pairs contributing to the extractor's knowledge are

$$\mathbf{q}^{(1)} M = (\mu_1^{(1)}, \ldots, \mu_s^{(1)}) \qquad \ldots \qquad \mathbf{q}^{(t)} M = (\mu_1^{(t)}, \ldots, \mu_s^{(t)}) \ ,$$

or $VM = W$, where V is the $t \times n$ matrix whose rows are $\{\mathbf{q}^{(i)}\}$ and W is the $t \times s$ matrix whose rows are $(\mu_1^{(i)}, \ldots, \mu_s^{(i)})$. The row-reduced echelon matrix A is related to V by $A = UV$, where U is a $t \times t$ matrix with nonzero determinant computed in applying Gaussian elimination to V.

The extractor's knowledge is initially empty, i.e., $\mathbb{D} = \emptyset$.

The extractor repeats the following behavior until $\#(\text{free}\,\mathbb{D}) \geq \rho n$:

The extractor chooses a random query Q. It runs $\mathcal{B}$ on Q. Suppose $\mathcal{B}$ chooses to respond, giving answer $(\mu_1, \ldots, \mu_s)$; clearly this happens with probability ϵ. Let Q be over indices $I \in [1, n]$, and denote it in vector notation as $\mathbf{q}$. Now we classify Q into three types:

1. $\mathbf{q} \notin \mathbb{D}$;
2. $\mathbf{q} \in \mathbb{D}$ but $I \not\subseteq \text{free}\,\mathbb{D}$; or
3. $I \subseteq \text{free}\,\mathbb{D}$.

For queries of the first type, the extractor adds Q to its knowledge $\mathbb{D}$, obtaining new knowledge $\mathbb{D}'$, as follows. It adds a row corresponding to the query to V, obtaining V', and a row corresponding to the response to W, obtaining W'; it modifies the transform matrix U, obtaining U', so that $A' = U'V'$ is again in row-reduced echelon form and spans $\mathbf{q}$. The primed versions $\mathbb{D}'$, A', U', V', and W' replace the unprimed versions in the extractor's state. For queries of type 2 or 3, the extractor does not add to its knowledge. Regardless, the extractor continues with another query.

Clearly, a type-1 query increases $\dim \mathbb{D}$ by 1. If $\dim \mathbb{D}$ equals n then $\text{free}\,\mathbb{D} = [1, n]$ and $\#(\text{free}\,\mathbb{D}) = n \geq \rho n$, so the extractor's query phase is guaranteed to terminate by the time it has encountered n type-1 queries.

We now observe that any time the simulator is in its query phase, type-1 queries make up at least a $1 - \omega$ fraction of the query space. By Claim A.1, type-2 queries make up at most a $(1/\#B)$ fraction of the query space, since

$$\begin{aligned}
\Pr_Q\big[Q \text{ is type-2}\big] &= \Pr_Q\big[\mathbf{q} \in \mathbb{D} \wedge I \not\subseteq \text{free}\,\mathbb{D}\big] \\
&= \Pr_Q\big[\mathbf{q} \in \mathbb{D} \;\big|\; I \not\subseteq \text{free}\,\mathbb{D}\big] \cdot \Pr_Q\big[I \not\subseteq \text{free}\,\mathbb{D}\big] \\
&\leq \Pr_Q\big[\mathbf{q} \in \mathbb{D} \;\big|\; I \not\subseteq \text{free}\,\mathbb{D}\big] \\
&\leq 1/\#B \ ,
\end{aligned}$$

where it is the last inequality that follows from the claim.[6] Here the probability expressions are all over a random choice of query Q, and I and $\mathbf{q}$ are the index set and vector form corresponding to the chosen query.

Similarly, suppose that $\#(\text{free}\,\mathbb{D}) = m$. Then by Claim A.2, type-3 queries make up at most an $m^l/(n - l + 1)^l$ fraction of the query space, and since $m < \rho n$ (otherwise the extractor would have ended the query phas) this fraction is at most $(\rho n)^l/(n - l + 1)^l$.

Therefore the fraction of the query space consisting of type-1 and type-2 queries is at most $1/\#B + (\rho n)^l/(n - l + 1)^l = \omega$. Since query type depends on the query and not on the randomness supplied to $\mathcal{B}$, it follows that the fraction of query-and-randomness-tape space consisting of type-1

[6] The claim gives a condition for a single I satisfying the condition $I \not\subseteq \text{free}\,\mathbb{D}$; the inequality here is over *all* such I; but if the probability never exceeds $1/\#B$ for any specific I then it doesn't exceed $1/\#B$ over a random choice of I, either.

and type-2 queries is also at most ω. Now, $\mathcal{B}$ must respond correctly on an ϵ fraction of the query-and-randomness-tape space. Even if the adversary is as unhelpful as it can be and this ϵ fraction includes the entire ω fraction of type-2 and type-3 queries, there remains at least an $(\epsilon-\omega)$ fraction of the query-and-randomness-tape space to which the adversary will respond correctly and in which the query is of type 1 and therefore helpful to the extractor. (By assumption $\epsilon > \omega$, so this fraction is nonempty.)

Since the extractor needs at most n successful type-1 queries to complete the query phase and it obtains a successful type-1 query from an interaction with $\mathcal{B}$ with probability $O\big(1/(\epsilon-\omega)\big)$, it follows that the extractor will require at most $O\big(n/(\epsilon-\omega)\big)$ interactions.

With $\mathbb{D}$ represented by a basis matrix A in row-reduced echelon form, it is possible, given a query $\mathbf{q}$ to which the adversary has responded, to determine efficienty which type it is. The extractor appends $\mathbf{q}$ to A and runs the Gaussian elimination algorithm on the new row, a process that takes $O(n^2)$ time [7, Section 2.3].[7] If the reduced row is not all zeros then the query is type 1; the reduction also means that the augmented matrix A' is again in row-reduced echelon form, and the steps of the reduction also give the appropriate updates to the transform matrix U'. Since the reduction need only be performed for the ϵ fraction of queries to which $\mathcal{B}$ correctly responds, the overall running time of the query phase is $O\big((1+\epsilon n^2)(n) \;/\; (\epsilon-\omega)\big)$.

Once the query phase is complete, the extractor has matrices A, U, V, and W such that $VM = W$ (where $M = (m_{ij})$ is the matrix consisting of encoded file blocks), $A = UV$, and A is in row-reduced echelon form. Moreover, there are at least ρn free dimensions in the subspace $\mathbb{D}$ spanned by A and by V. Suppose i is in free $\mathbb{D}$. Since A is in row-reduced echelon form, there must be a row in A, say row t, that equals the ith basis vector $\mathbf{u}_i$. Multiplying both sides of $VM = W$ by U on the left gives the equation $AM = UW$. For any j, $1 \le j \le s$, consider the entry at row t and column j in the matrix AM. It is equal to $\mathbf{u}_i \cdot (m_{1,j}, m_{2,j}, \ldots, m_{n,j}) = m_{i,j}$. If we compute the matrix product UW, we can thus read off from it every block of every sector for $i \in$ free $\mathbb{D}$. Computing the matrix multiplication takes $O(n^2 s)$ time. The extractor computes the relevant rows, outputs them, and halts. □

B Erasure Codes

It is easier to verify that a server is storing *half* the blocks of a file — or any other constant fraction r — than to verify that it is storing *all* the blocks of a file: probabilistic checks are unlikely to uncover a single file block's being dropped. Before storing it on the server, we would therefore like to encode an n-block file into a $2n$-block file — or, more generally (n/r)-block file — with the encoding done in such a way that any n blocks suffice for recovering the original file.

Erasure codes are the codes that provide this property [23, 2]. The parameter r is called the rate. Some erasure codes are *rateless* in that, for an n-block file, they allow the generation of arbitrarily many blocks, any n of which suffice for decoding.[8]

An important property of erasure codes is the efficiency of their encoding and decoding procedures. Ideally, one would like both procedures to have performance linear in n. This is especially important for our application, where n can be very large. For example, if a block is 1000 bytes and

[7] More specifically, $O(tn)$ time if A is a $t \times n$ matrix; but of course $t \le n$.

[8] Some erasure codes, called *nearly-optimal*, require somewhat more than n blocks for decoding: $n(1+\epsilon)$ blocks, where ϵ is a parameter; a choice of ϵ has an effect on other code parameters.

the file being stored is 1 GB, then we have $n \approx 2^{20}$. A code where encoding and decoding take $O(n^2)$ time would be unpleasantly slow.

Another important property of erasure codes is what sort of erasures they can correct. Ideally, we would like the code to correct against arbitrary erasures: *any* n blocks should suffice for recovering the original file. Some codes, however, correct only against random erasures: any n blocks suffice for decoding with overwhelming probability, whereas an adversarially selected n block set will not suffice.

Unfortunately, no codes are known that provide linear decoding time in the presence of arbitrary erasure. As we show below, we can still make use of codes that correct random erasures only, but in this case additional secret preprocessing is required that makes public retrievability impossible.

B.1 Codes for Public Retrievability

Traditional Reed-Solomon–style erasure codes can be constructed for arbitrary rates allowing recovery of the original file from any r fraction of the encoded file blocks [24]. The encoding and decoding procedures will take $O(n^2)$ time. The code matrix used can be made public and any user can apply the decoding procedure. This provides public retrievability.

We recommend that a *systematic* code be used: one in which the first m blocks of the encoded file are in fact the encoded file itself. This makes recovering the file from a server that isn't malicious much more efficient: simply ask for the first m blocks.

B.2 Efficient Codes for Private Retrievability

To employ codes with linear-time encoding and decoding procedures, we will need to take additional measures to transform adversarial erasure to random erasure.

Given a file M, encrypt M using a semantically-secure symmetric encryption scheme under a key k_{enc1}. Now apply the erasure code to the encrypted, permuted file. Assuming the encryption was secure, this is indistinguishable (to any efficient adversary) from applying the erasure code to a random bitstring of the same length as M.

Each block in the erasure-encoded file M^* depends on some subset of the blocks in the unencoded, encrypted file. Which blocks, precisely, depends on the code family and the particular key used to instantiate a code from that family. Let the unencoded file consist of blocks with indices in the range $[1, m]$, and the encoded file of blocks with indices in the range $[1, n]$. For $i \in [1, n]$, let $\pi_k(i) \subseteq [1, m]$ be the pre-encoding blocks on which encoded block i depends for a choice k of code key. Suppose that the erasure code is such that:

1. The distribution of $\pi_k(i)$ is independent of the distributions for $\pi_k(j)$ for all $j \neq i$. More precisely, the conditional distribution of $\{\pi_k(i)\}_k$ over keys k satisfying a specific assignment for $\pi_k(j)$ for all $j \neq i$ is statistically indistinguishable from the distribution $\{\pi_k(i)\}_k$ over all keys k.

2. Each block in the encoded file depends on all blocks in the unencoded file with equal probability. More precisely, for all $a, b \in [1, m]$ and for all $i \in [1, n]$, $\Pr_k[a \in \pi_k(i)] = \Pr_k[b \in \pi_k(i)]$.

Such a code would necessarily have a long key, but that key can be generated from a short seed using a secure pseudorandom generator. In fact, it is easy to see that the two properties above are provided by Online Codes [21, 18], a simple nearly-optimal linear-time code.

Now, for any code that provides the two properties above, an adversary could determine which blocks in the encoded file depend on which blocks in the unencoded file only using the contents of the encoded blocks. This is because, ignoring the block contents,

Let the encoded blocks be $\eta_1, \ldots, \eta_n$. We encrypt each block independently under key k_{enc2}, using a tweakable block cipher [20], with the tweak for the encryption of η_i being the block index i; the blocks output by this procedure, $m_1, \ldots, m_n$ are those stored on the server. Assuming the cipher is secure, the contents of these blocks are indistinguishable from random and independent of each other.

It is now clear that, without knowledge of k_{enc1}, k, or k_{enc2}, the encoded file $m_1, \ldots, m_n$ reveals no information about the code used, and, specifically, about which code blocks depend on which pre-encoding blocks. Thus no adversary that erases blocks can do better than random erasure, which is exactly the property we require for decoding to work with overwhelming probability.

C Can We Eliminate B Coefficients?

In both schemes proposed in Section 3, the verifier sends with each index i of a query a coefficient ν_i from a set B. If we could avoid sending these coefficients — equivalently, if we could set $B = \{1\}$, then we would obtain a scheme that is more efficient in several respects:

- the verifier would need to flip fewer coins in generating a query;
- the query would be shorter by $l \cdot \lg \#B$ bits;
- the prover's computation would be greatly reduced: essentially, one multiplication instead of $l + 1$ multiplications in the first scheme; one exponentiation instead of $l + 1$ exponentiations in the second scheme[9]; and
- the verifier's computation would also be reduced, though not so dramatically.

Unfortunately, it is clear from Lemma 4.5 that the proof techniques of Section 4.2 cannot apply when $\#B = 1$, since we will not then have $\epsilon < 1/\#B$, however large the adversary's success probability ϵ is.

This is not just a proof problem. Below, we present an attack on the schemes of Section 3 when $B = \{1\}$. In the attack, the server stores $n - 1$ blocks instead of n, can answer a nonnegligible fraction of all queries, yet no extraction technique can recover *any* of the original blocks.

A note on notation. In this section, we will make some simplifications to the notation for the sake of brevity and clarity. First, obseve that the Part-1 proofs of Section 4.1 *do* apply in the case that $B = \{1\}$. We will thus elide the authenticators $\{\sigma_i\}$ in our attack; this allows us to address both the scheme with private verification and the scheme with public verification simultaneously. Second, we will set the number of sectors per block, s, to 1. Our attack easily generalizes to the case $s > 1$, but this simplification allows us to eliminate a subscript and simplify the presentation.

[9] The l element summation computed by the prover in the first scheme requires work comparable to a multiplication when $l \approx \lg p$.

The attack. With the simplifications above, consider an n-block file with blocks $(m_1, \ldots, m_n)$. A query will consist of l indices $I \subset [1, n]$; the response will be $\mu = \sum_{i \in I} m_i$. We assume that l is even.[10]

The adverary chooses an index i^* at random from $[1, n]$. For each $i \neq i^*$, he chooses $\tau_i \xleftarrow{R} \{-1, +1\}$ and sets

$$m'_i \leftarrow m_i + \tau_i m_{i^*} \ .$$

Now the adversary remembers $(m'_1, \ldots, m'_{i^*-1}, m'_{i^*+1}, m'_n)$. Clearly, the adversary needs to store one less block than an honest server would.

Now, consider a query I. If $i^* \notin I$, the adversary responds with $\mu' = \sum_{i \in I} m'_i$. Otherwise, $i^* \in I$, and the adversary responds with $\mu' = \sum_{i \in I \setminus \{i^*\}} m'_i$.

In our analysis, we will use the following simple lemma:

Lemma C.1 ([1], p. 12)**.** *For $k \geq 2$ we have $\binom{k}{\lfloor k/2 \rfloor} \geq 2^k/k$.*

Proof. $\binom{k}{\lfloor k/2 \rfloor}$ is the largest of the k values $\binom{k}{1}$, $\binom{k}{2}$, $\ldots$, $\binom{k}{k-1}$, and $\binom{k}{0} + \binom{k}{l}$; and so it must be at least as large as their average, $2^k/k$. □

In the case $i^* \notin I$, μ' will be correct provided that we have $\sum_{i \in I} \tau_i = 0$. But this happens when the number of $+1$s and the number of -1s are equal in $\{\tau_i\}_{i \in I}$, and this happens with probability

$$\binom{l}{l/2} \cdot \left(\frac{1}{2}\right)^{l/2} \cdot \left(\frac{1}{2}\right)^{l/2} \geq \left(\frac{2^l}{l}\right) \cdot \left(\frac{1}{2^l}\right) = \frac{1}{l} \ .$$

In the case $i^* \in I$, μ' will be correct provided we have $\sum_{i \in I \setminus \{i^*\}} \tau_i = 1$. This happens when there are $l/2 - 1$ -1s and l $+1$s in $\{\tau_i\}_{i \in I \setminus \{i^*\}}$, and this happens with probability

$$\binom{l-1}{l/2-1} \cdot \left(\frac{1}{2}\right)^{l/2-1} \cdot \left(\frac{1}{2}\right)^{l/2} = \binom{l-1}{\lfloor (l-1)/2 \rfloor} \cdot \left(\frac{1}{2}\right)^{l-1} \geq \left(\frac{2^{l-1}}{l-1}\right) \cdot \left(\frac{1}{2^{l-1}}\right) = \frac{1}{l-1} > \frac{1}{l} \ .$$

Thus the adversary can respond to $1/l$ fraction of queries where $i^* \in I$ and to a $1/l$ fraction of queries where $i^* \notin I$; so he can respond to a $1/l$ fraction of *all* queries, which is clearly nonnegligible.

But now it is impossible for any extraction strategy to recover *any* block, let alone a ρ fraction of all blocks. This is because the subspace known to the adverary is insufficient to determine any block. Indeed, the adversary's knowledge is consistent with any value for any block. Fix $(m'_1, \ldots, m'_{i^*-1}, m'_{i^*+1}, m'_n)$ where $m'_i = m_i + \tau_i m_{i^*}$. Suppose we believe that $m_{i^*} = a$ for any value a. This fixes m_i for each $i \neq i^*$, as $m_i = m'_i - \tau_i m_{i^*}$. If we believe, for some index $\lambda \neq i^*$, $m_\lambda = a$, then m_{i^*} is fixed because $m'_\lambda = m_\lambda + \tau_\lambda m_{i^*}$ implies $m_{i^*} = (\tau_\lambda)(m'_\lambda - a)$, and the argument proceeds as before. Since the adversary's knowledge is consistent with any choice of value for any (single) block, it cannot be the case that it allows recovery of the value of any block.

[10] If l is odd, the adversary can set $m'_i \leftarrow m_i + (l^{-1} \bmod p)(m_{i^*})$, which allows him to respond to that l/n fraction of queries where $i^* \in I$.

D Construction with RSA Signatures

D.1 Construction

Let λ_1 be a bitlength such that the difficulty of factoring a $(2\lambda_1 - 1)$-bit modulus is appropriate to the security parameter. Let $\max B$ be the largest element in B, and let λ_2 be a bitlength equal to $\lceil \lg(l \cdot \max B) \rceil$.

The construction of the private verification scheme PubRSA is:

PubRSA.Kg(). Generate a random signing keypair $(spk, ssk) \xleftarrow{R} \mathsf{SKg}$. Choose two random primes p' and q' in the range $\left[2^{\lambda_1 - 2}, 2^{\lambda_1 - 1} - 1\right]$, where p' and q' are, additionally, Sophie Germain primes, so that $p = 2p' + 1$ and $q = 2q' + 1$ are also prime. Let $N = pq$ be the RSA modulus; we have $2^{2\lambda_1 - 2} < N < 2^{2\lambda_1}$. The group of quadratic residues of N, QR_N, has order $p'q'$. Let $H\colon \{0,1\}^* \to QR_N$ be a full-domain hash, which we treat as a random oracle. Choose a random $(2\lambda_1 + \lambda_2)$-bit prime e, and set $d = e^{-1} \bmod \phi(N)$. The secret key is $sk = (N, d, H, ssk)$; the public key is $pk = (N, e, H, spk)$.

PubRSA.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: $\{m_{ij}\}_{\substack{1 \le i \le n \\ 1 \le j \le s}}$. Each sector m_{ij} is an element of $\mathbb{Z}_N$. Now parse sk as (N, d, H, ssk). Choose a random file name *name* from some sufficiently large domain (e.g., $\mathbb{Z}_N$). Choose s random elements $u_1, \ldots, u_s \xleftarrow{R} QR_N$. 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: $t \leftarrow t_0 \| \mathsf{SSig}_{ssk}(t_0)$.

Now, for each i, $1 \le i \le n$, compute

$$\sigma_i \leftarrow \left(H(name \| i) \cdot \prod_{j=1}^{s} u_j^{m_{ij}} \right)^d \bmod N \ .$$

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

PubRSA.$\mathcal{V}(pk, sk, t)$**.** Parse pk as (N, e, H, 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 $\mu_1, \ldots, \mu_s$ and $\sigma \in \mathbb{Z}_N$. Check that each μ_j is in the range $[0,\ l \cdot N \cdot \max B]$. If parsing fails or the $\{\mu_j\}$ values are not in range, fail by emitting 0 and halting. Otherwise, check whether

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

if so, output 1; otherwise, output 0.

PubRSA.$\mathcal{P}(pk, t, M^*)$**.** Parse the processed file M^* as $\{m_{ij}\}$, $1 \le i \le n$, $1 \le j \le s$, along with $\{\sigma_1\}$, $1 \le i \le n$. 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 $\nu_i \in B$.

For each j, $1 \leq s \leq j$, compute

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

where the sum is computed in $\mathbb{Z}$, without modular reduction. In addition, compute

$$\sigma \leftarrow \prod_{(i,\nu_i)\in Q} \sigma_i^{\nu_i} \bmod N \ .$$

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

D.2 Part-One Proof

We now give the part one proof of our scheme.

We will use the following lemma (see [15] and Lemma 1 of [10]):

Lemma D.1. *Given $x, y \in \mathbb{Z}_N$, along with $a, b \in \mathbb{Z}$ such that $x^a = y^b$ and $\gcd(a, b) = 1$, one can efficiently compute $\bar{x} \in \mathbb{Z}_N$ such that $\bar{x}^a = y$.*

Theorem D.2. *If the signature scheme used for file tags is existentially unforgeable and the RSA problem with large public exponents is hard, 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 $\mathsf{PubRSA}.\mathcal{P}$.*

Once more, we prove the theorem as a series of games.

Game 0. The first game, Game 0, is simply the challenge game defined in Section 2. By assumption, the adversary $\mathcal{A}$ wins with nonnegligible probability.

Game 1. Game 1 is the same as Game 0, with one difference. The challenger keeps a list of all signed tags ever issued as part of a store-protocol query. If the adversary ever submits a tag t either in initiating a proof-of-storage protocol or as the challenge tag, that (1) has a valid signature under *ssk* but (2) is not a tag signed by the challenger, the challenger declares failure and aborts.

Clearly, if there is a difference in the adversary's success probability between Games 0 and 1, we can use the adversary to construct a forger against the signature scheme.

Game 2. Game 2 is the same as Game 1, with one difference. The challenger keeps a list of its responses to St queries made by the adversary. Now the challenger observes each instance of the proof-of-storage protocol with the adversary — whether because of a proof-of-storage query made by the adversary, or in the test made of $\mathcal{P}'$, or as part of the extraction attempt by Extr. If in any of these instances the adversary is successful (i.e., $\mathcal{V}$ outputs 1) but either

1. the adversary's aggregate signature σ is not equal to $\prod_{(i,\nu_i)\in Q} \sigma_i^{\nu_i} \bmod N$ (where Q is the challenge issued by the verifier and σ_i are the signatures on the blocks of the file considered in the protocol instance) or

2. at least one the adversary's aggregate block values $\mu'_1, \ldots, \mu'_s$ is not equal to the expected block value $\mu_j = \sum_{(i,\nu_i)\in Q} \nu_i m_{ij}$,

the challenger declares failure and aborts.

Before analyzing the difference in success probabilities between Games 1 and 2, we will establish some notation and draw a few conclusions. Suppose the file that causes the abort is n blocks long, has name *name*, has generating exponents $\{u_j\}$, and contains sectors $\{m_{ij}\}$, and that the block signatures issued by St are $\{\sigma_i\}$. Suppose $Q = \{(i, \nu_i)\}$ is the query that causes the challenger to abort, and that the adversary's response to that query was $\mu'_1, \ldots, \mu'_s$ together with σ'. Let the expected response — i.e., the one that would have been obtained from an honest prover — be $\mu_1, \ldots, \mu_s$ and σ, where $\sigma = \prod_{(i,\nu_i)\in Q} \sigma_i^{\nu_i} \bmod N$ and $\mu_j = \sum_{(i,\nu_i)\in Q} \nu_i m_{ij}$ for $1 \le j \le s$. By the correctness of the scheme, we know that the expected response satisfies the verification equation, i.e., that

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

Because the challenger aborted, we know that σ' and $\mu'_1, \ldots, \mu'_s$ passed the verification equation, i.e., that

$$(\sigma')^e = \prod_{(i,\nu_i)\in Q} H(name\|i)^{\nu_i} \cdot \prod_{j=1}^{s} u_j^{\mu'_j} \ .$$

Now observe that condition 1, above, implies condition 2, which means that having the simulator abort on either condition 1 or 2 is the same as having abort on just condition 2: if condition 2 doesn't hold then $\mu'_j = \mu_j$ for each j, and it follows from the verification equation that $(\sigma')^e = \sigma^e$; because e is relatively prime to $\phi(N)$, this means that $\sigma' = \sigma \mod N$, and since $\mathcal{V}$ checked that σ' is in $\mathbb{Z}_N$, this means that $\sigma' = \sigma$, so condition 1 doesn't hold, either.

Therefore, if we define $\Delta\mu_j \stackrel{\text{def}}{=} \mu'_j - \mu_j$ for $1 \le j \le s$, it must be the case that if the simulator aborts at least one of $\{\Delta\mu_j\}$ is nonzero.

With this in mind, we now show that if there is a nonnegligible difference in the adversary's success probabilities between Games 1 and 2 we can construct a simulator that solves the RSA problem when the public exponent e is large.

The simulator is given as inputs a $2\lambda_1$-bit modulus N and a $(2\lambda_1 + \lambda_2)$-bit public exponent e, along with a value $y \in QR_N$; its goal is to output $x \in QR_N$ such that $x^e = y$. The simulator behaves like the Game 1 challenger, with the following differences:

- In generating a public key, it sets the modulus and public exponent to N and e; it does not know the corresponding secret modulus d.

- The simulator programs the random oracle H. It chooses a random generator g of QR_N. It keeps a list of queries and responses to answers consistently. In answering the adversary's queries it chooses a random $r \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_{N^2}$ and responds with $g^r \bmod N$. (The larger space $\mathbb{Z}_{N^2}$ means that the distribution of $\{g^r\}$ is statistically indistinguishable from QR_N, even though the simulator does not know the order of QR_N because it doesn't know $\phi(N)$.) The simulator also answers queries of the form $H(name\|i)$ in a special way, as we will see below.

- When asked to store some file whose coded representation comprises the n blocks $\{m_{ij}\}$, $1 \le i \le n$, $1 \le j \le s$, the simulator behaves as follows. It chooses a name *name* at random.

Because the space from which names are drawn is large, it follows that, except with negligible probability, the simulator has not chosen this name before for some other file and a query has not been made to the random oracle at $name\|i$ for any i.

For each j, $1 \le j \le s$, the simulator chooses random values $\beta_j, \gamma_j \xleftarrow{R} \mathbb{Z}_{N^2}$ and sets $u_j \leftarrow g^{e \cdot \beta_j} y^{\gamma_j}$. For each i, $1 \le i \le n$, the simulator chooses a random value $r_i \xleftarrow{R} \mathbb{Z}_{N^2}$, and programs the random oracle at i as

$$H(name\|i) = g^{e \cdot r_i} \Big/ \prod_{j=1}^{s} u_j^{m_{ij}} \ .$$

Now the simulator can compute σ_i, since we have

$$H(name\|i) \cdot \prod_{j=1}^{s} u_j^{m_{ij}} = g^{e \cdot r_i} \ ;$$

if the simulator sets $\sigma_i = g^{r_i}$, we will have $\sigma_i^e = g^{e \cdot r_i} = \big(H(name\|i) \cdot \prod_{j=1}^{s} u_j^{m_{ij}}\big)^{\alpha}$, as required.

- The simulator continues interacting with the adversary until the condition specified in the definition of Game 2 occurs: the adversary, as part of a proof-of-storage protocol, succeeds in responding with a signature σ' that is different from the expected signature σ.

  The change made from Game 0 to Game 1 establishes that the parameters associated with this protocol instance — $name$, n, $\{u_j\}$, $\{m_{ij}\}$, and $\{\sigma_i\}$ — were generated by the simulator as part of a St query; otherwise, execution would have already aborted. This means that these parameters were generated according to the simulator's procedure described above. Now, dividing the verification equation for the forged signature σ' by the verification equation for the expected signature σ, we obtain

$$(\sigma'/\sigma)^e = \prod_{j=1}^{s} u_j^{\Delta\mu_j} = g^{e \cdot \sum_{j=1}^{s} \beta_j \Delta\mu_j} \cdot y^{\sum_{j=1}^{s} \gamma_j \Delta\mu_j} \ ;$$

  rearranging terms yields

$$\Big[(\sigma'/\sigma) \cdot g^{\sum_{j=1}^{s} \beta_j \Delta\mu_j}\Big]^e = y^{\sum_{j=1}^{s} \gamma_j \Delta\mu_j} \ ; \tag{2}$$

  Now, provided that $\gcd\big(e, \sum_{j=1}^{s} \beta_j \Delta\mu_j\big) = 1$, we can compute, using Lemma D.1, a value x from (2) such that $x^e = y$. It remains only to argue that $\gcd\big(e, \sum_{j=1}^{s} \beta_j \Delta\mu_j\big) \neq 1$ with negligible probability. First, we noted already that not all of $\{\Delta\mu_j\}$ can be zero. Second, the values of $\{\gamma_j\}$ are statistically hidden from the adversary,[11]. Thus, provided that $\gcd(e, \Delta\mu_j) = 1$ for each j where $\Delta\mu_j \neq 0$, we will have $\gcd\big(e, \sum_{j=1}^{s} \beta_j \Delta\mu_j\big) = 1$ except with probability $1/e$, which is negligible. But for any j where $\Delta\mu_j \neq 0$ we have $\mu_j, \mu'_j \in [0, l \cdot N \cdot \max B]$, so

$$|\Delta\mu_j| = |\mu'_j - \mu_j| \le l \cdot N \cdot \max B < 2^{\lceil \lg N \rceil} \cdot 2^{\lceil \lg(l \cdot \max B) \rceil} < 2^{2\lambda_1} \cdot 2^{\lambda_2} < e \ ,$$

  and since e is prime this means that $\gcd(\Delta\mu_j, e)$ must equal 1.

Thus if there is a nonnegligible difference between the adversary's probabilities of success in Games 1 and 2, we can construct a simulator that uses the adversary to solve the RSA problem, as required.

[11] Hidden because they are used to compute only the values $\{u_j\}$ in the adversary's view, and these are Pedersen commitments with the blinding factors $g^{e\beta_j}$ are drawn statistically close to the uniform distribution on QR_N.

Wrapping up. Assuming the signature scheme used for file tags is secure, and that the RSA problem with large public exponent is hard, we see that any adversary that wins the soundness game against our public-verification scheme responds in proof-of-storage protocol instance with values $\{\mu_j\}$ and σ that are computed according to $\mathsf{Pub}.\mathcal{P}$, which completes the proof of Theorem D.2. □

We also note that our random oracle programming techniques have a similar flavor to that of Ateniese et al. [4, 3].

D.3 Part-Two and Part Three Proofs

It is easy to see that the Part-Two proof of Section 4.2 carries over unchanged to the case where blocks are drawn from $\mathbb{Z}_N$ instead of $\mathbb{Z}_p$. All the matrix operations require is that inversion be efficiently computable, and this is, of course, the case in $\mathbb{Z}_N$ using Euclid's algorithm, provided we never encounter values in $\mathbb{Z}_N \setminus \mathbb{Z}_N^*$; but such a value would allow us to factor N, so they occur with negligible probability provided the RSA problem — and therefore factoring — is hard.

Similarly, erasure decoding works just as well when blocks are drawn from $\mathbb{Z}_N$; and because nothing in the proof requires that blocks be distributed uniformly in all of $\mathbb{Z}_N$, we could treat each m_{ij} as an element of $\mathbb{Z}_{p_0}^t$ where p_0 is some prime convenient for whatever erasure code we employ and t is the largest integer such that $p_0^t < N$.

E Extensions

In this section we discuss a few possible extensions to our scheme.

Towards adaptively adding storage blocks. In our current scheme the storage algorithm takes as input a whole file M before storing the data and publishing the random *name* value. In some situations it might make sense to be able to incrementally append blocks to the end of the file. For example, we might want to append to the file itself.

Our current proof relies upon the simulator being able to "program" the random oracle so that it can sign the blocks of M. The simulator doesn't not give out the *name* value until it learns the whole file M value so that the attacker cannot query the random oracle until the simulator knows how to program it. However, if we want to allow for appending the simulator will need to disclose *name* before learning some values of the block values.

We can overcome this issue by adding a bit of randomness to our system. In addition, to storing s sectors per data block the storage algorithm will additionally store one more element that is chosen randomly in $\mathbb{Z}_p$. The storage, proof, and verification algorithms will behave just as though there were $s + 1$ sectors stored. In the simulation, the simulator will set the program the oracle so that it can sign a "random" block , and when it gets a request to store a specific block it will choose the randomness of the last "dummy" sector such that it can sign the entire block. Our techniques are similar to those used by Coron [8] for tight proofs of signature schemes.

We note that these techniques are useful for validating the integrity of blocks appended to the end, however, in a complete scheme we still need to concern ourselves with how to redundantly encode the file such that blocks can be appended to the end.

Toward removing random oracles. One interesting question is whether we can realize our scheme in the standard model (without random oracles). One step towards this direction is to

use a common random string consisting of n_{Max} group elements $h_1, \ldots, h_{n_{\text{Max}}}$, where n_{Max} is the maximum number of blocks used in a file. In this situation, we could replace the call to $H(name\|i)$ with h_i. In addition, the simulator would use the techniques from the paragraph above so that it could "program" the h values for signing a "random" message and later use the extra sector to make the proof go through.