Proofs of Replicated Storage Without Timing Assumptions*

Ivan Damgård, Chaya Ganesh, and Claudio Orlandi

{ivan,ganesh,orlandi}@cs.au.dk,
Department of Computer Science, DIGIT, Aarhus University

Abstract

In this paper we provide a formal treatment of *proof of replicated storage*, a novel cryptographic primitive recently proposed in the context of a novel cryptocurrency, namely Filecoin.

In a nutshell, proofs of replicated storage is a solution to the following problem: A user stores a file m on n different servers to ensure that the file will be available even if some of the servers fail. Using proof of retrievability, the user could check that every server is indeed storing the file. However, what if the servers collude and, in order to save on resources, decide to only store one copy of the file? A proof of replicated storage guarantees that, unless the server is indeed reserving the space necessary to store n copies of the file, the user will not accept the proof. While some candidate proofs of replicated storage have already been proposed, their soundness relies on timing assumptions i.e., the user must reject the proof if the prover does not reply within a certain time-bound.

In this paper we provide the first construction of a proof of replication which does not rely on any timing assumptions.

1 Introduction

Consider a scenario where a user A wants to use the cloud or some other decentralized network of servers to store and distribute some file m to other users. To make sure she and other users will be able to access the file later on, A stores several replicas of m in different locations. However, A suspects that the servers she is using are adversarial and may collude, for instance to save on costs by using less space than they are supposed to. So she will be interested in checking that indeed unique space has been dedicated to each replica, and it is natural to require that this can be verified, even if all servers are controlled by an adversary. We will call this *proof of replication*.

A first issue to note is that the well-known notions of proof of retrievability or proof of space (which we discuss in more detail below) do not solve the problem if each replica is simply a copy of m. Such proofs allow a user to check that a given file is retrievable from a server, much more efficiently than by simply retrieving the file. However, even if A asks for a proof of retrievability of m from each of the servers and all these proofs are successful, this may simply be because the user is actually talking to the adversary who stores only a single copy of m.

*This work was supported by: the Protocol Labs RFP Program; the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreements No 669255 (MPCPRO) and No 803096 (SPEC); the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC); the Concordium Blockchain Research Center, Aarhus University, Denmark.

Another idea that comes to mind is that A could let each replica be an encryption of m under some key K, but with fresh randomness for each replica. If the encryption is IND-CPA secure, the adversary cannot distinguish this from encryptions of random independent messages, and hence it seems he is forced to store all replicas in order for them to be retrievable later. While this intuition can in fact be proved, this would not be a satisfactory solution: recall that we want that anyone, not just A, can retrieve the original file, so A would have to share K with other users. However, if any of these users collude with the adversary, the security breaks down. Besides, a solution that does not require A to store secret information for later is clearly more practical.

The idea of proof of replication was introduced in Filecoin [Lab17a, Lab17b], a decentralized storage network[1]. They articulate a list of properties that they desired from such a notion. They define a *Sybil attack* which is exactly what we discussed above: if an honest client wishes to store the same file m on n different servers, an adversary can store these using sybil identities (all servers are controlled by one adversary) and successfully pass the storage audit, while essentially storing only one copy of the file.

A decentralized storage network defined in [Lab17a], as an abstraction is a network of independent storage providers that offer verifiable file storage and retrieval services. In the Filecoin protocol, miners earn protocol tokens by providing data storage services. In such a scenario, it is crucial that any security model allow the adversary to choose the file m. This is because an adversary could request for m to be stored, and then prove that m was stored to collect network rewards. One may consider such an attack in a case where the client is honest and is fooled into storing a particular file by the adversary, or one can consider a corrupt user working with a set of corrupted servers. The latter is what is referred to as the *generation attack* in the Filecoin paper, where the adversary can simply determine m such that it can be efficiently regenerated on demand.

While the Filecoin paper does not give a formal treatment of proof of replication, they propose a construction for what they call a time-bounded proof of replication. In such a notion, the file to be stored is encoded so that the encoding process is slow: slow enough for a client to distinguish between honest proving time, and potentially adversarial proving time which includes the time to re-encode. Thus, the encoding process is, by design, distinguishably more expensive than honest proving time. This notion is realized by using a block-cipher and slowing it down by block chaining. A time-bounded proof of replication is a proof of storage of a replica that is encoded in this way. Even if the proof of storage scheme used offers public verifiability, this time-bounded proof of replication is publicly verifiable only if the encoding key (or the original file) is made publicly available by the client, or by computing the encodings within a scheme that proves computational integrity and privacy, for instance, a SNARK (Succinct Non-interactive ARguments of Knowledge). As discussed earlier, a solution that does not need to store any secret information is desirable, and using generic SNARKs would be computationally expensive.

In any case, the basic problem with all time-bounded schemes is the handling of recomputing attack: the encoding has to be made so slow that even a powerful server cannot encode faster than the time a proof takes. This is harder than it may seem at first: even if we know for sure how many operations are needed to encode for a given value of security parameter, the actual time it takes depends on the hardware held by the adversarial parties, and so is beyond the control of honest users. This makes a concrete choice of parameters very difficult: should we compensate for the adversary being more powerful than we expect and choose a very slow encoding thus making life

[1] Other related notions in the context of data replication have been studied earlier in the cryptographic literature; we discuss the connection and differences in the related work section.

harder also for honest clients when they encode? Or should we choose parameters more aggressively and run a bigger risk of being cheated? It would clearly be better if we did not have to make such a choice.

We ask if we can do better in all the above aspects: can we have a proof of replication scheme that provably resists sybil attacks, offers public verifiability and is not time-bounded?

Our Results. We give a formal treatment of proofs of replication, by giving a definition that captures the desired properties as well as a construction which we prove secure according to the definition. The construction works in the random oracle model and can be instantiated from any one-way permutation.

Each replica of the file m to be stored in our construction has size $O(|m| + \kappa)$, where $|m|$ is the length of m and κ is the security parameter. To verify replication, the user conducts a proof of retrievability with each server. Any such proof can be used, so we inherit whatever communication complexity that proof has.

We concentrate on the case where the client doing the encoding is honest, as this seems to be the most important case in practice, and is in line with the definitions of proof of retrievability and storage. But we also give at the end of the paper two solutions that work for a corrupt client, at the expense of using more communication.

Very roughly speaking, the idea in our solution is that the adversary first receives each of the replicas to store, where each replica is a special encoding of m. He may now store a state for later use, which in the honest case would contain all replicas. What we show is that, no matter how the adversary computes the state, if it is significantly smaller than the combined size of all replicas, then some of the proofs of retrievability will fail, unless the adversary breaks a computational assumption.

Let us consider what this exactly guarantees us. Since the proofs of retrievability are *extractable*, the above guarantees that the replicas cannot be compressed i.e., the adversary must reserve enough space to store all replicas, and this space must contain some data which is equivalent (up to polynomial time computation) to the replicas of the file. But is this the best we can do? Why don't we ask that the adversary in fact stores a concatenation of all the replicas? Actually, this goal is impossible to achieve: even honest servers will most likely store the same information in different formats (think of *little-* vs. *big-endian* representation). So we certainly cannot expect that the adversary will store exactly the same data that was received from the client. However, this should not matter from a security point of view, as long as the original data can be efficiently recovered.

In conclusion: it is impossible to force the corrupted servers to store *exactly* the n replicas or n copies of the file. Therefore, the best we can hope for is what we do in this paper: no matter how the corrupted servers behave, it is possible to recover n distinct, incompressible encodings of the same file, thus the servers cannot pass the verification unless they reserve the necessary space for all replicas. Note, however, that in such a scenario, the corrupted servers have no incentive to do anything else than simply store the replicas.

The main difference between our work and previously proposed solutions to the problem, is that our solution does not require the use of *time*: while the original, informal definition of proof of replication states that it should be hard for the server to recompute the encodings of the file in the time it takes to verify the proof, our definition is much stronger as it rules out that any polynomially bounded attacker who uses less storage than claimed can pass the verification. As discussed above, this makes implementing proof of replications much easier, since one does not need to worry about finding an appropriate value for the verifier timeout.

To avoid misunderstandings, we emphasize that even if our definitions and proof work with a single adversary that handles all replicas, the actual use case includes several servers that each store one replica (if they are honest). Since it is clearly impossible to check if a server stores something without talking to that server, the communication complexity of our protocol must be proportional to the number of servers[2].

1.1 Related Work

Proofs of retrievability. A lot of user data today is outsourced for storage on the cloud both because of large volumes of data, and for reliability in case of failure of local storage. The problem with cloud storage is that of maintaining integrity of data and enforcing accountability of the storage provider. Proofs of retrievability, first formalized by Juels and Kaliski in [JK07] address this problem by allowing for audits. In a proof of retrievability, a client can store a file on the server, while storing (a short) verification string locally. In an audit protocol, the client acts as the verifier and the server proves that it possesses the client's file. The property that the server "possesses" a file is formalized by the existence of an extractor that retrieves the client's file from a server that makes a client accept in the audit protocol. Since their introduction, there have been several works [SW08, DVW09] constructing proof of retrievability schemes with a proof of security and efficient audit procedures. One property we prioritize in this work is *public verifiability* where any party can take the role of the verifier in the audit protocol, not just the client who originally stored the file. This means the client's state storing any verification information for the file should not contain any secrets. The construction of [SW08] gives a proof of retrievability scheme secure in the random oracle model that allows public verifiability.

Proofs of space. A proof of space is a protocol where a prover convinces a verifier that it has dedicated a significant amount of disk-space. Proofs of space were introduced in [DFKP15] as an alternative to proof of work (PoW), and further studied in [RD16, AAC$^+$17]. There have been proposals based on proof of space like chia network [chi17] and Spacemint [PPK$^+$15]. Very roughly, a proof of space gives the guarantee that it is more "expensive" for a malicious server that dedicates less space than an honest server to successfully pass an audit.

Data replication. Curtmola et al. [CKBA08] and Barsoum et al. [BH11] propose protocols that enable proofs of data replication in the private verifier model, where the client stores a secret key that is used for verification. The work of Hao and Yu [HY10] allows public verifiability but nevertheless requires the client to store a secret. The work of Etemad and Küpçü [EK13] studies replicated provable data possession, but does not formalize replicated storage, and the client need not be aware of any replication. Finally, the protocol of Armknecht et al. [ABBK16] is also in the private verification model, and in addition, uses RSA time-lock puzzles which results in a protocol with a time-bounded property that we elaborate on below.

[2] Of course, if a single server would store all replicas, we can optimize the communication needed, this is also easy to see for our protocol, but this hardly seems like an interesting use case.

Time-bounded Proofs of Replication. In a recent work by Pietrzak [Pie18], a construction for proof of replication based on proof of space is given. A proof of replication is not formally defined, and therefore it is not clear what is the replication property that the construction satisfies. In addition, since a proof of space is the starting point of the construction, it has the same "time-bounded" property as the Filecoin construction, since a malicious server can pass the audit by recomputing data. More recently, [FBBG18, BBBF18, CFMJ18, Fis18] construct proofs of replication based on slow encodings. They have the same time-bounded flavour of other recent works and is thus significantly different from ours.

1.2 Technical Overview

The existing time-bounded proofs use a public deterministic encoding function. The problem is that this always allow a malicious server to recompute encoded data and this may lead to a successful recomputation attack if the server has sufficient computational resources. Our observation is that one can instead make the encoding be probabilistic. Now the adversary will only see the encoded data but not the randomness that the client used to encode. One may therefore hope that recomputing an encoding is not only slow, but completely unfeasible. On the other hand, decoding must still be easy for anyone.

To illustrate the idea of our solution, we start with a toy example: we assume that we are given oracle access to a random permutation T, and its inverse[3], acting on strings $\{0,1\}^n$. As is well known (and discussed in detail later) we can instantiate such an oracle in the standard random oracle model. In order to create replicas of a file, A will generate an instance of a one-way trapdoor permutation $f : \{0,1\}^n \mapsto \{0,1\}^n$, with trapdoor t_f. For simplicity, we assume that the file m to store is an $(n - \log n)$-bit string. Then the i'th replica is defined to be $(f, f^{-1}(T(m||i)))$, where $||$ denotes concatenation and f is a specification of the 1-way permutation. Clearly, anyone can easily compute m from a replica by computing f in the forward direction and calling T^{-1}. It turns out that this construction is secure if the adversary computes the state to store for later in a very restricted way, namely he forgets all information about at least one replica, say the i'th one. Namely, the adversary forgets both the encoding $(f, f^{-1}(T(m||i)))$ and the intermediate value $T(m||i)$.

We can now argue that if the adversary is nevertheless able to produce the i'th replica, he will have to invert the one-way permutation: from the output of the adversary $(f, f^{-1}(T(m||i)))$ we can (as the encoding can be decoded efficiently), extract $T(m||i)$. But, we assumed that the state did not contain any information about this value (except for a negligible amount following from the fact that it must be different from other outputs). Hence he must call the oracle to get $T(m||i)$. Therefore, in a security reduction, we can take a challenge value y and reprogram T such that $T(m||i) = y$. Now, the i'th replica (that we assumed the adversary could produce) is exactly the preimage of y under f.

Of course, we cannot reasonably assume that the adversary behaves in this simple-minded way. As mentioned, we only want to assume that the state stored is smaller than the combined size of the replicas, say by a constant factor. To overcome this problem, we iterate the above construction several times, so that T is called several times while preparing a replica. Now there are many more outputs from T than the adversary can remember, and we show that by the setting the parameters right, at least one of these is almost uniform in the view of the adversary. Now we can place a

[3] One can think of the random permutation T as a random oracle which can be invoked in both directions

challenge value for the one-way permutation in this position by an argument similar to the above.

2 Preliminaries

Notation. We denote the concatenation of two bit strings x and y by $x||y$. Throughout, we use κ to denote the security parameter. We denote a probabilistic polynomial time algorithm by PPT. A function is negligible if for all large enough values of the input, it is smaller than the inverse of any polynomial. We use negl to denote a negligible function. We use $[1, n]$ to represent the set of numbers $\{1, 2, \ldots, n\}$. For a randomized algorithm Alg, we use $y \leftarrow \mathsf{Alg}(x)$ to denote that y is the output of Alg on x. We write $y \stackrel{R}{\leftarrow} \mathcal{Y}$ to mean sampling a value y uniformly from the set $\mathcal{Y}$.

2.1 Trapdoor permutations

A collection of trapdoor permutations is a family $\mathcal{F} = \{f_{pk} : \mathcal{D}_{pk} \to \mathcal{D}_{pk}\}$ such that:

- There exists a PPT algorithm KeyGen such that $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa)$, f_{pk} is a permutation.
- There exists a PPT algorithm that given pk samples uniformly from $\mathcal{D}_{pk}$.
- There exists a PPT algorithm that on input pk and $x \in \mathcal{D}_{pk}$, computes $f_{pk}(x)$.
- There exists a PPT algorithm that on input sk and $f_{pk}(x)$, computes x, that is, $f_{sk}^{-1}(f_{pk}(x)) = x$.

Definition 1. *A trapdoor permutation family $\mathcal{F} = \{f_{pk} : \mathcal{D}_{pk} \to \mathcal{D}_{pk}\}$ is said to be hard to invert if the following holds: for all PPT algorithms A, there exists a negligible function* negl *such that*

$$\Pr[f_{pk}(z) = y : (pk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa), x \leftarrow \mathcal{D}_{pk}, y \leftarrow f_{pk}(x), z \leftarrow A(pk, y)] \leq \mathsf{negl}(\kappa)$$

When the domain and range is clear from context, we omit the subscript pk and only write $\mathcal{D}$.

Definition 2. *We call a trapdoor permutation family a B-leakage trapdoor permutation if the following holds: For all PPT algorithms (A_1, A_2), there exists a negligible function* negl *such that*

$$\Pr[f_{pk}(z) = y : (pk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa), x \leftarrow \mathcal{D}_{pk}, L \leftarrow A_1(x, pk), y \leftarrow f_{pk}(x), z \leftarrow A_2(y, L(x))] \leq \mathsf{negl}(\kappa)$$

where the output length of L is bounded by B bits.

Note that every trapdoor function family is also a B-leakage trapdoor permutation family for $B = \log \kappa$.

RSA trapdoor permutation. The RSA trapdoor permutation is given by:

- $\mathsf{KeyGen}(1^\kappa)$: Choose κ-bit primes p, q, let $N = pq$. Choose e such that $\gcd(e, (p-1)(q-1)) = 1$, let d be such that $ed = 1 \bmod (p-1)(q-1)$. Return $(pk = (e, N), sk = d)$
- For $x \in \mathbb{Z}_N^*$, given $pk = (e, N)$, compute $f_{pk}(x) = y = x^e \bmod N$.
- For $y \in \mathbb{Z}_N^*$, and $sk = d$, compute $f_{sk}^{-1}(y) = y^d \bmod N$

The RSA inversion problem is assumed to be hard for any $\mathcal{A}$ running in time polynomial in κ.

Invertible Random Oracle. We assume the algorithms of the construction and the adversary have access to an invertible random oracle (IRO): that is oracle access to $\Pi : \mathcal{D} \rightarrow \mathcal{D}$ and $\Pi^{-1} : \mathcal{D} \rightarrow \mathcal{D}$. We require that Π be indifferentiable from a random permutation. The indifferentiablity framework, first proposed by Maurer et al. [MRH04], informally says that given ideal primitives G and F, a construction C^G is indifferentiable from F, if there exists a simulator S with oracle access to F such that (C^G, G) is indistinguishable from (F, S^F). Coron et al. [CHK+16] showed that a 14-round Feistel network where the round functions are independent random oracles is indifferentiable from a random permutation. A series of subsequent works [DKT16, DS16] show that 8 rounds is sufficient.

On instantiating the oracles. In our constructions, we make use of an invertible random oracle H that acts on strings of arbitrary length, and an invertible random oracle T that has the same domain as a trapdoor permutation. H is instantiated using a regular Feistel network. In the following we discuss how to instantiate T on pairs of outputs of RSA and obtain trapdoor permutation f and IRO T with the same domain.

We define a trapdoor permutation $f : (\mathbb{Z}_N)^2 \rightarrow (\mathbb{Z}_N)^2$ as follows: $f(x_1, x_2) = (f'(x_1), f'(x_2))$ where f' is the RSA permutation $f' : \mathbb{Z}_N \rightarrow \mathbb{Z}_N$. Note that N is part of the public key of the RSA permutation. The input and output of T are elements in $(\mathbb{Z}_N)^2$. We note that we can instantiate the Feistel construction in this domain as well by replacing XOR with multiplication modulo N i.e., given a random oracle G that maps inputs in $\mathbb{Z}_N^*$ to strings that are twice the length, we can define $\mathcal{F} : (\mathbb{Z}_N)^2 \rightarrow (\mathbb{Z}_N)^2$ on pairs of values modulo N as follows:

$$\mathcal{F}^H(L||R) = s||t, \text{ where } s = L \cdot G(R) \bmod N, t = R \cdot G(s) \bmod N$$

where $\cdot$ is product modulo N. Note that $G(x) \bmod N$ is close to uniform in $\mathbb{Z}_N^*$, therefore, $\mathcal{F}$ is invertible except with negligible probability i.e., if $\mathcal{F}$ is not invertible then a non-trivial factor of N is found.

2.2 Proof of retrievability

Proofs of retrievability, introduced by Juels and Kaliski [JK07] allow a client to store data on a server that is untrusted, and admit an *audit* protocol in which the server proves to the client that it is still storing all of the data. A scheme without random oracle was given in [DVW09], whereas [SW08] allows public verifiability. A proof of retrievability (PoR) scheme consists of three algorithms, $\mathsf{Gen}, \mathcal{P}, \mathcal{V}$. We recall the definition from [SW08, DVW09] below.

- The generation algorithm takes as input a file $F \in \{0,1\}^*$ and outputs a file to be stored on the server and a tag (verification information) for the client.

$$(F^*, \tau) \leftarrow \mathsf{Gen}(F)$$

- The $\mathcal{P}, \mathcal{V}$ algorithms define an audit protocol to prove retrievability of the file. The $\mathcal{P}$ algorithm takes as input the processed file F^* and the $\mathcal{V}$ algorithm takes the tag τ. At the end of the audit protocol, the verifier outputs a bit indicating whether the proof succeeds or not.

$$\{0,1\} \leftarrow \langle \mathcal{P}(F^*), \mathcal{V}(\tau) \rangle$$

Experiment $\mathsf{Expt}_{\mathcal{A}}^{PoR\text{-}sound}(\kappa)$

- The adversary $\mathcal{A}$ picks a file $F \in \{0,1\}^n$.
- The challenger creates $(F^*, \tau) \leftarrow \mathsf{Gen}(F)$ and returns F^* to $\mathcal{A}$.
- $\mathcal{A}$ can interact with $\mathcal{V}(\tau)$ by running many proofs and seeing whether $\mathcal{V}$ outputs 0 or 1.
- $\mathcal{A}$ outputs a prover algorithm (ITM) $\mathcal{P}^*$ and returns this to the challenger.
- The challenger runs $b \leftarrow \langle \mathcal{P}^*, \mathcal{V}(\tau) \rangle$, and runs the extractor, $\tilde{F} = \mathsf{ext}^{\mathcal{P}^*}(\tau, n, \kappa)$
- Output 1 if $b = 1 \wedge \tilde{F} \neq F$, or 0 otherwise.

Figure 1: Soundness for Proofs of Retrievability.

A PoR scheme needs to satisfy correctness and soundness. Correctness requires that for all file $F \in \{0,1\}^*$, and for all (F^*, τ) output by $\mathsf{Gen}(F)$, an honest prover will make the verifier accept in the audit protocol.

$$\langle \mathcal{P}(F^*), \mathcal{V}(\tau) \rangle = 1$$

Informally, a PoR scheme is sound if for any prover that convinces the verifier that it is storing the file, there exists an algorithm called the extractor that interacts with the prover and extracts the file. We give the formal definition below.

Definition 3 (Soundness for Proof of Retrievability)**.** *A proof of retrievability (PoR)* $\mathsf{Gen}, \mathcal{P}, \mathcal{V}$ *satisfies* soundness *if for any PPT adversary* $\mathcal{A}$*, there exists an extractor* ext *such that the advantage of* $\mathcal{A}$

$$\mathsf{Adv}_{\mathcal{A}}^{PoR\text{-}Sound}(\kappa) = \Pr[\mathsf{Expt}_{\mathcal{A}}^{PoR\text{-}Sound}(\kappa) = 1]$$

in the experiment described in Figure 1 is negligible in κ*.*

The definition in [DVW09] discusses the notion of knowledge soundness versus information soundness. If the definition holds for the class of efficient extractors, the scheme satisfies knowledge soundness. A somewhat weaker notion is that of information soundness where the running time of the extractor is not restricted.

2.3 Min entropy

Recall that the predictability of a random variable X is $\max_x \Pr[X = x]$ and its min-entropy $\mathsf{H}_\infty(X)$ is $-\log(\max_x \Pr[X = x])$. The average case min-entropy is defined as follows. Let X and Y be random variables.

$$\tilde{\mathsf{H}}_\infty(X|Y) = -\log\left(\mathbb{E}_{y \leftarrow Y}\left(2^{-\mathsf{H}_\infty(X|Y=y)}\right)\right)$$

We make use of the following lemma which states that the average min-entropy of a variable (from the point of view of an adversary) does not go down by more than the number of bits (correlated with the variable) observed by the adversary. We recall the entropy weak chain rule for average case min entropy below in Lemma 1.

Lemma 1. *([DORS08]) Let X and Y be random variables. If Y has at most 2^λ values, then*

$$\tilde{\mathsf{H}}_\infty(X|Y) \geq \mathsf{H}_\infty(X) - \mathsf{H}_0(Y) = \mathsf{H}_\infty(X) - \lambda$$

where $\mathsf{H}_0(Y) = \log|\mathsf{support}(Y)|$

3 Defining Proof of Replication

While several candidates of proof of replication have already been proposed, we are not aware of any formal definition of the security properties that such a proof should satisfy. It is indeed non-trivial to come up with the "right" definition, due to the fact that we ask the adversary to store many copies *of the same file*. Thus simply requiring the existence of an extractor algorithm (as in proof of knowledge or proof of storage) is not sufficient: it is not enough that the adversary knows the file, the adversary should know multiple replicas of the same file. But what does it mean for an extractor to extract replicas of the same file? Before providing our definition, we introduce some notions of encodings which will be used to build up our solution.

3.1 Replica Encodings

We now define $\mathsf{ReplicaEncoding}$ as a tuple of algorithms ($\mathsf{rEnc}, \mathsf{rDec}$) where rEnc takes a message $m \in \{0,1\}^*$ and outputs a replica encoding of $m \in \{0,1\}^*$,

$$y \leftarrow \mathsf{rEnc}(\kappa, m)$$

The rDec algorithm takes a replica encoding and returns a message i.e., $m \leftarrow \mathsf{rDec}(y)$.

Definition 4 (Replica encoding)**.** *A pair* ($\mathsf{rEnc}, \mathsf{rDec}$) *is a secure replica encoding if the following holds:*

- *Completeness: The probability of incorrect decoding is negligible i.e.,*

$$\Pr[\mathsf{rDec}(\mathsf{rEnc}(\kappa, m)) \neq m] < \mathsf{negl}(\kappa)$$

- *Soundness: Consider the game* $\mathsf{sound}_{\mathcal{A}_1,\mathcal{A}_2}$ *between an adversary and a challenger defined in Figure 2. A replica encoding scheme is c-sound (for a constant $c, 0 < c < 1$) if for any* $(\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that the following holds.*

$$\Pr\left[|\mathsf{state}| < cv\beta | v \leftarrow \mathsf{sound}_{\mathcal{A}_1,\mathcal{A}_2}\right] \leq \mathsf{negl}(\kappa)$$

  where β is the bit-length of an encoding y.

- *Efficiency:* $|y| = |m| + O(\kappa)$.

Experiment $\mathsf{sound}_{\mathcal{A}_1,\mathcal{A}_2}$

- The adversary $\mathcal{A}_1$ chooses a file $m \in \{0,1\}^k$
- The challenger outputs n encodings of m

$$y^{(i)} \leftarrow \mathsf{rEnc}(\kappa, m)$$

  for $i \in [1, n]$ and returns $(y^{(1)}, \ldots, y^{(n)})$ to $\mathcal{A}_1$.
- $\mathcal{A}_1$ outputs a state $\mathsf{state} \leftarrow \mathcal{A}_1(y^{(1)}, \ldots, y^{(n)})$
- The challenger runs $\mathcal{A}_2$ on state.

$$(\tilde{y}^{(1)}, \ldots, \tilde{y}^{(n)}) \leftarrow \mathcal{A}_2(\kappa, \mathsf{state})$$

- Let $v_i = 1$ if $\tilde{y}^{(i)} = y^{(i)}$, and 0 otherwise. Output $v = \sum_{i=1}^{n} v_i$.

Figure 2: Soundness of a Replica Encoding scheme

Discussion. The main measure of efficiency for a replica encoding is its expansion factor, in other words the ratio $|y|/|m|$. Clearly, the smaller the expansion factor the more interesting the scheme is. Looking ahead, all our constructions will have $|y| = |m| + O(\kappa)$.

We motivate here some of the choices in our definition. First note that the completeness requirement allows the file to be reconstructed from a single replica encoding. This captures the functional requirement in the honest usage of proofs of replication, where a client would store different encodings of the file on different servers and should be able to recover the file as long as one server is storing their encoding.

When defining soundness, we consider a monolithic adversary $\mathcal{A}$ which controls all colluding servers. To be able to meaningfully talk about the *space* that the adversary uses for storing the file, we split the adversary $\mathcal{A}$ into two parts $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A}_1$ receives the replica encoding from the challenger (representing the honest client) and $\mathcal{A}_2$ is the part of the adversary returning (some of) the encodings to the client at a later stage, using the state that $\mathcal{A}_1$ transferred to $\mathcal{A}_2$. We do not require that $\mathcal{A}_2$ outputs all of the received encoding, instead, we use the variable v to count how many of the replica encodings $\mathcal{A}_2$ is able to return. The definition of soundness then intuitively states that the adversary can at most return the number of replicas that "fit" into the state (where we allow for a constant "slack" c, to avoid trivial attacks where the adversary forgets few bits and then "guesses" them right before returning the encodings – in practice one should think of c as any constant close to 1).

As a sanity check of our definition, let's consider a construction of replica encoding that is "too trivial": define $y^{(i)} = (m, r_i)$ i.e., every replica is simply the message concatenated with some random string r_i. Due to incompressibility of random data the adversary needs to store all the r_i's, but clearly only needs to store one copy of m and can still recompute all encodings. This is of course not desirable, so our definition had better not accept this construction. Indeed it does not: the adversary can break the soundness property because he can choose to return $v \geq 2$ encodings using storage only $|m| + v|r| < cv|y|$ - which is trivially true for any interesting case (remember in efficient encodings $|y| \approx |m|$ and c is close to 1).

3.2 Proof of Replication

We now use the notion of encodings to meaningfully capture the replication property. A proof of replication scheme consists of a tuple of algorithms $\mathsf{create}, \mathsf{retrieve}$ and an audit protocol defined by two algorithms, $\mathcal{P}, \mathcal{V}$ for the prover and verifier respectively. create is a randomized algorithm that takes as input a file $m \in \{0,1\}^*$, that is to be replicated and stored, a replication factor n; and produces n replicas $y^{(1)}, \ldots, y^{(n)}$ together with verification information ver. Each replica $y^{(i)}$ is sent the server i to be stored, and ver with the client to be used for verification in the audit protocol. $\mathsf{retrieve}$ is a deterministic algorithm run by anyone that takes as input a replica $y^{(i)}$ and outputs a file m^*.

In the audit protocol, each server (prover) has a replica $y^{(i)}$, and the client (verifier) has ver. At the end of the audit, the verifier outputs a bit b indicating whether the audit was successful or not. We denote the protocol executing the prover and verifier algorithms by $\langle \mathcal{P}_i(\tilde{y}^{(i)}), \mathcal{V}(\mathsf{ver}, i) \rangle$.

We require the scheme to satisfy completeness and soundness properties as defined below. Note that when considering the honest usage of the our protocol (e.g., completeness), each server is able to prove to the client that they are storing the file independently[4]. On the other hand, when considering adversarial behaviour (e.g., soundness), we assume that all servers are under the control of a monolithic adversary.

All our algorithms below are parametrized by a security parameter, even when omitted as in the description below.

Definition 5 (Proof of Replication)**.** *A scheme* $\mathsf{PoRep} = (\mathsf{create}, \mathsf{retrieve}, \mathcal{P}, \mathcal{V})$ *where,*

$$(y^{(1)}, \ldots, y^{(n)}, \mathsf{ver}) \leftarrow \mathsf{create}(m, n), \ \textit{for } m \in \{0,1\}^*, n \in \mathbb{Z}$$

$$m_i^* = \mathsf{retrieve}(y^{(i)}), i \in [1, n]$$

$$\{0,1\} \leftarrow \langle \mathcal{P}_i(\tilde{y}^{(i)}), \mathcal{V}(\mathsf{ver}, i) \rangle$$

is a proof of replication scheme if the following properties are satisfied.

- *Completeness. For an honest client and honest server,*
  - *for* $(y^{(1)}, \ldots, y^{(n)}, \mathsf{ver}) \leftarrow \mathsf{create}(m, n), m_i^* = \mathsf{retrieve}(y^{(i)}), m_i^* = m \ \forall i \in [1, n]$
  - *The audit protocol interaction between honest client and honest server succeeds, that is, the client accepts and outputs* $b = 1$.

$$\langle \mathcal{P}_i(\tilde{y}^{(i)}), \mathcal{V}(\mathsf{ver}, i) \rangle = 1$$

- *Soundness. We define the soundness game* $\mathsf{sound}^{\mathcal{E}}_{\mathcal{A}_1, \mathcal{A}_2}$ *between an adversary and a challenger in Figure 3. The scheme* PoRep *is c-sound (for a constant* $c, 0 < c < 1$*) if for any* $(\mathcal{A}_1, \mathcal{A}_2)$*, there exists an extractor* $\mathcal{E}$ *and a negligible function* negl *such that the following holds.*

$$\Pr\left[u < v \vee |\mathsf{state}| < cv\beta | (u, v) \leftarrow \mathsf{sound}^{\mathcal{E}}_{\mathcal{A}_1, \mathcal{A}_2}\right] \leq \mathsf{negl}(\kappa)$$

Experiment $\mathsf{sound}^{\mathcal{E}}_{\mathcal{A}_1,\mathcal{A}_2}$

- The adversary $\mathcal{A}_1$ chooses a file $m \in \{0,1\}^k$
- The challenger runs $(y^{(1)}, \ldots, y^{(n)}, \mathsf{ver}) \leftarrow \mathsf{create}(m, n)$ and returns $(y^{(1)}, \ldots, y^{(n)})$ to $\mathcal{A}_1$.
- $\mathcal{A}_1$ outputs a state $\mathsf{state} \leftarrow \mathcal{A}_1(y^{(1)}, \ldots, y^{(n)})$
- The challenger runs $\langle A_2(\mathsf{state}), \mathcal{V}(\mathsf{ver}, i)\rangle$, let v_i be the output of $\mathcal{V}$ for all $i \in [1, n]$ and $v = \sum_{i=1}^{n} v_i$.
- The challenger runs the extractor.

$$(\tilde{y}^{(1)}, \ldots, \tilde{y}^{(n)}) = \mathcal{E}^{A_2}(\kappa, \mathsf{ver}, k)$$

- For all $i \in [1, n]$, define $u_i = 1$ if $\tilde{y}^{(i)} = y^{(i)}$, and $u = \sum_{i=1}^{n} u_i$
  The output of the game $\mathsf{sound}^{\mathcal{E}}_{\mathcal{A}_1,\mathcal{A}_2}$ is (u, v).

Figure 3: Soundness of a proof of replication scheme

where β is the bitlength of an encoding y.

The definition above guarantees that the malicious servers, even colluding, cannot make the verifier accept more proofs than the storage they have used.

4 Constructing Proof of Replication

We begin by giving a high-level overview of our construction. Following the idea behind our definition, we create many independent encodings, and use a proof of retrievability on the encodings. Even though each encoding can independently be decoded to the same file without any secret information, the proof of retrievability on the encodings enforces that the server stores each encoding and therefore dedicates space for *each* replica. Recall that in Section 1.2 we have already described a simple solution which works in a restricted model in which the adversary is restricted to either store or delete entire replicas. Of course this is not a realistic threat model and a malicious server could choose to forget arbitrary parts of each encoding (say, a constant fraction). Now, to pass the audit, the server would have to compute a preimage of the underlying trapdoor permutation, but *given* a constant fraction of bits of the preimage. Unfortunately, the definition of security for trapdoor permutation does not allow us to say that this is not possible; in other words, we cannot construct a reduction for this kind of adversaries.

To address this problem, we use the following approach: we start by applying an (invertible) random oracle (IRO) on the message concatenated with a short seed (which is different for each

[4] For instance, an honest server does not need to communicate with the other servers, nor know that they exist.

replica), and then we use the trapdoor permutation on the result. We then iterate the IRO and the trapdoor permutation as a round function sufficiently many times. Intuitively the trapdoor permutation of the round function ensures that the adversary has to do something "hard" in every round, while the IRO of the round function is used to make sure that the "hard tasks" are all independent.

Again, in any given round, we cannot rule out that the adversary might have stored some (small) information that allows to easily invert the trapdoor permutation. However, since we repeat this for many rounds and the adversary must store some pre-image information at every round to potentially break the trapdoor permutation, eventually the total information that the adversary would have to store will exceed the bound necessary for replicated storage.

When dealing with large files (e.g., larger than the size of the input/output of T), we split the file in blocks. To make sure that all blocks depend on the entire file (for instance, to prevent the server from "de-duplicating" individual blocks which might appear in multiple files), we first apply a "large" IRO on the entire file. Then, in the round function, we apply a "small" IRO on each individual block. Thanks to this, the number of rounds in the encoding only needs to be proportional to the block size, instead of the entire file size (as it was the case in an earlier version of this paper).

Note that our combination of the RSA trapdoor permutation with a random oracle is reminiscent of full domain hash-signatures and, to a greater extent, CCA secure encryption via RSA and OAEP. Note however that, for our proof to go through, we need T to be indifferentiable from a random oracle, thus the two Feistel rounds of OAEP are not enough. Moreover, in our construction, we apply the oracle and the trapdoor permutation for multiple rounds, and the domain of the random oracle is a pair of blocks for the RSA permutation. The idea of iterating a combination of RSA with a random oracle was used before in [VDJO$^+$12], however (apart from their work having a less in-depth treatment) there are two major differences, namely that they did not consider replication as an application, and that they use a strictly weaker notion of security, namely "near-incompressibility".

Efficiency of decoding. We note that, when instantiating the construction with the RSA trapdoor permutation, it is possible to use a small exponent (i.e., $e = 3$). Now decoding would be much faster then encoding, which is a desirable property in applications where a single user uploads a file which is then retrieved by a large number of users.

4.1 Replica Encodings

We now proceed to describe our construction in detail, and first construct a replica encoding scheme $\mathsf{ReplicaEncoding} = (\mathsf{rEnc}, \mathsf{rDec})$ in Figures 4 and 5.

Soundness of the scheme. Before formally proving the soundness of the scheme, we give an overview of the proof idea. If the state state that is passed from $\mathcal{A}_1$ to $\mathcal{A}_2$ is small, then the adversary $\mathcal{A}_2$ cannot "remember" all the answers to the queries that $\mathcal{A}_1$ made in the first part of the game since the outputs of a random oracle are incompressible. However, we can extract the outputs of the random oracle used during the encoding from the adversary since the replica encodings can be efficiently decoded. This implies that the adversary must make some queries to the random oracle in phase 2 of the game. Now, for each of the queries the adversary makes, there are two options: either the response to the query has full entropy in the view of the adversary, or it

Let $m \in \{0,1\}^{k'}$ be a message to be encoded.

- Choose a string r uniformly at random from $\{0,1\}^{\kappa}$, and let $y_0 = H(m||\mathsf{r})$, where $H : \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$ is a invertible random oracle (IRO), and $\lambda = k' + \kappa$.
- Let $(\mathsf{KeyGen}, f^{-1}, f)$ be trapdoor permutation over domain $\mathcal{D}$. $(sk, pk) \leftarrow \mathsf{KeyGen}(1^{\kappa})$. Divide y_0 into s blocks such that each block is in $\mathcal{D}$. That is, $y_0 = Y_{10}|| \cdots ||Y_{s0}$. Let $T : \mathcal{D} \to \mathcal{D}$ be an IRO over $\mathcal{D}$. We then iterate the following round function: For each round j from 1 to r, and for each block $t \in [1, s]$ define
  - Apply the IRO T,
  $$Z_{tj} = T(Y_{tj-1})$$
  - Invert the trapdoor permutation block-wise,
  $$Y_{tj} = f_{sk}^{-1}(Z_{tj})$$

  Let $y_j = Y_{1j}|| \cdots ||Y_{sj}$
- Let $R = (y_r, pk)$
- Return R

Figure 4: The Replica Encoding Algorithm $\mathsf{rEnc}(\kappa, m)$

For a replica $R = (y_r, pk)$, Parse y_r as $Y_{1r}|| \cdots ||Y_{sr}$. For each round j from r down to 1, and each block $t \in [1, s]$, compute

- Round j:
  - Apply the trapdoor permutation block-wise,
  $$Z_{tj} = f_{pk}(Y_{tj})$$
  - Invert the IRO,
  $$Y_{tj-1} = T^{-1}(Z_{tj})$$
- Let $y_0 = Y_{10}|| \cdots ||Y_{s0}$. Compute $H^{-1}(y_0)$ and parse the output as $m||\mathsf{r}$ where m is the first k' bits. Return m.

Figure 5: The replica decoding algorithm $\mathsf{rDec}(R)$

doesn't. If it has full entropy, i.e., if the state contained no (or very little) information about what the oracle would answer, then we are done, as we will elaborate next. But first let us consider if not; that is, the response to the query made did not have full entropy in $\mathcal{A}_2$'s view. This means that the state must have contained some information about the answer to the query. Now, since the encoding uses the random oracle in each round, and since the state that the adversary is allowed to remember is small, by carefully accounting for the entropy budget for each query made, we argue that after a certain round, the entropy in the state is exhausted and therefore there is at least one query that the adversary had to make to the oracle whose response has full entropy in order to win the game. Finally, once we have found such a query for which we know that the output of the oracle has full entropy that $\mathcal{A}_2$ *had to make* to win the game, we can reprogram the random oracle with a challenge for the trapdoor permutation. Thus since $\mathcal{A}_2$ is nevertheless able to produce the replicas, we use it to break the assumption and reach a contradiction.

Theorem 1. *Assuming T, H are invertible random oracles, the construction* $\mathsf{ReplicaEncoding} = (\mathsf{rEnc}, \mathsf{rDec})$ *using trapdoor permutation f is a secure replica encoding scheme, replication parameter n as per Definition 4. For number of rounds $r > \frac{(cn+1)k}{B}$, it is complete and c-sound with soundness error*

$$\epsilon \leq \left(\epsilon' + 2^{-k(1-c)} + qs^2 2^{-k}\right) nrs$$

where $k = \log|\mathcal{D}|$, $\mathcal{D}$ is the domain of f and T, s is the number of blocks, q the number of queries to the RO and the advantage of any adversary in B-leakage inversion of the permutation f is at most ϵ'.

Proof. We use the notation $R^{(i)}$ to identify the encoding stored on server i.

Completeness. For n encodings that are created honestly, $R^{(i)} \leftarrow \mathsf{rEnc}(m, \kappa)$, $m^* = \mathsf{rDec}(R^{(i)})$. Then, due to the invertibility of T, H and the trapdoor permutation f, $m^* = m$, $\forall i$.

Soundness. Assume there exists an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ such that

$$\Pr\left[|\mathsf{state}| < cv\beta | v \leftarrow \mathsf{sound}_{\mathcal{A}_1,\mathcal{A}_2}\right] > \epsilon \ .$$

Therefore, the adversary $\mathcal{A}_2^T$ outputs $R^{(i_1)}, \cdots, R^{(i_v)}$, where each $i_j \in [1, n]$. Let $I \subset [1, n], |I| = v$ be the set of indices indicating the replicas that $\mathcal{A}_2$ outputs correctly. We argue that if the state is too small, then the adversary does not have enough entropy to store information about $R^{(i)}$, $\forall i \in I$ and therefore one of the $R^{(i)}$ must have been recomputed, (by making relevant queries to T), which we use to invert the B-leakage trapdoor permutation. The proof idea is that since the state is small, in round r, $\mathcal{A}_2$ must have learned some of the Z values of round r from responses of T. If the response of T to these queries do not have full entropy in $\mathcal{A}_2$'s view given the state, then this deficit must be accounted for in the size of the state. We continue this argument for every round going backwards from the last round, by reasoning about the set of relevant queries made in each round, and accounting for every query made that did not have entropy in $\mathcal{A}_2$'s view with B bits in the state. We then hit a round where the response of T for one of $\mathcal{A}_2$'s queries must have high enough entropy from $\mathcal{A}_2$'s point of view. Such a query is guaranteed to exist since the state size is used up after enough queries of the former kind. We use the response to this query to embed a challenge and invert the trapdoor permutation. We now proceed to give the reduction.

Let $\mathcal{B}$ be an adversary whose task is to invert the trapdoor permutation. $\mathcal{B}$ receives a challenge $(\hat{pk}, \hat{x})$, and wins if it outputs $\hat{y}$ such that $f_{\hat{pk}}(\hat{y}) = \hat{x}$. $\mathcal{B}$ interacts with $(\mathcal{A}_1, \mathcal{A}_2)$ in the soundness game. $\mathcal{B}$ receives a file $m \in \{0,1\}^{k'}$ from $\mathcal{A}_1$. $\mathcal{B}$ creates encoded replicas honestly, except for one of the replicas chosen at random (call its index $i^* \in [1, n]$), in which the challenge will be embedded. For now we only use the public key $\hat{pk}$. Since $\mathcal{B}$ does not know the corresponding secret key, $\mathcal{B}$ cannot compute this encoding honestly. Thus, $\mathcal{B}$ defines the encoding $R^{(i^*)}$ as $(\hat{pk}, y_r)$ for a uniformly random y_r. Then it "decodes" y_r down to y_0 (by following the decoding procedure) and finally programs the random oracle H such that $H^{-1}(y_0) = (m||\mathsf{r}^{(i^*)})$ for some random string $\mathsf{r}^{(i^*)}$. More in detail:

- Choose random $y_r^{(i^*)} \in \{0,1\}^\lambda$.
- For each round j from r down to 1, parse $y_j^{(i^*)} = Y_{1j}||\cdots||Y_{sj}$ and, for each block $t \in [1, s]$, compute:
  - Apply the trapdoor permutation block-wise,
  $$Z_{tj} = f_{\hat{pk}}(Y_{tj})$$
  - Invert the IRO,
  $$Y_{tj-1} = T^{-1}(Z_{tj})$$
- Let $y_0 = Y_{10}||\cdots||Y_{s0}$.
- Pick a random value $\mathsf{r} \in \{0,1\}^\kappa$ and program the IRO H to output y_0 on input (m, r).
- Return $R^{(i^*)} = (y^{(i^*)}, \hat{pk})$

$\mathcal{B}$ responds to any other oracle queries of $\mathcal{A}_1$ honestly, and finally gives $(R^{(1)}, \cdots, R^{(n)})$ to $\mathcal{A}_1$, where $R^{(i)}$ for $i \neq i^*$ is created honestly. $\mathcal{A}_1$ outputs a state state. Now, $\mathcal{B}$ interacts with $\mathcal{A}_2$. It runs $\mathcal{A}_2$ on state, and receives and responds to $\mathcal{A}_2$'s oracle queries in the following way. $\mathcal{B}$ chooses a random round $j^* \in [1, r]$, and a random block $t^* \in [1, s]$ to embed the challenge $\hat{x}$ in. If $\mathcal{A}_2$ queries T on $Y_{t^*j^*-1}^{(i^*)}$, $\mathcal{B}$ sets the response to embed its challenge $\hat{x}$ in the following way. Set $Z_{t^*j^*} = \hat{x}$ and,

$$T(Y_{t^*j^*-1}^{(i^*)}) = Z_{t^*j^*}$$

The rest of the queries are answered honestly. If $\mathcal{A}_2$ makes a query $Y_{t^*j^*}^{(i^*)} = \hat{y}$ such that, $f_{\hat{pk}}(\hat{y}) = \hat{x}$, $\mathcal{B}$ outputs $\hat{y}$. If there is no such query, $\mathcal{B}$ outputs $\perp$. We now compute the probability that $\mathcal{B}$ wins in the trapdoor permutation inversion game. Consider the case when $(m||\mathsf{r})$ fits into $\mathcal{D}$, and therefore there is only one block in the encoding. We later show how the argument extends to multiple blocks. Let k be the block length, that is, the bit length of elements in $\mathcal{D}$. We therefore have $|\mathsf{state}| < cvk, c < 1$. Consider the min-entropy of the random variable state, which is at most the bit length, $\mathsf{H}_\infty(\mathsf{state}) \leq cvk$. $\mathcal{A}_2$ on state returns,

$$\{R^{(i)}\}_{i \in I} \leftarrow \mathcal{A}_2(\mathsf{state})$$

for $R^{(i)} = (y_r^{(i)}, pk_i)$. Since there is only one block in the encoding, $y_r^{(i)} \in \mathcal{D}$ and can be decoded to obtain $z_r^{(i)} = f_{pk_i}(y_r^{(i)})$.

Let $\mathcal{Y}_r = y_r^{(i_1)}|| \cdots ||y_r^{(i_v)}$ and $\mathcal{Z}_r = z_r^{(i_1)}|| \cdots ||z_r^{(i_v)}, i_j \in I$.

Since each replica is computed by using independent randomness r_i, the queries to the oracle are different for each replica, and therefore each $z_r^{(i)}$ is unpredictable. We have,

$$\mathsf{H}_\infty(z_r^{(i)}|z_r^{(1)}, \cdots, z_r^{(i-1)}, z_r^{(i+1)}, \cdots, z_r^{(n)}) = k$$

and therefore, $\mathsf{H}_\infty(\mathcal{Z}_r) = vk$. Since $\mathcal{Z}_r$ can be extracted from $\mathcal{A}_2^T(\mathsf{state})$ by decoding, either the state contains information about each $z^{(i)}$ in $z^{(1)}|| \cdots ||z^{(v)}$, or $\mathcal{A}_2$ must make relevant RO queries, that is, query the RO on the inputs corresponding to $z^{(i)}$. By the conditional rule for average case min-entropy (Lemma 1),

$$\tilde{\mathsf{H}}_\infty(\mathcal{Z}_r|\mathsf{state}) \geq \mathsf{H}_\infty(\mathcal{Z}_r) - \mathsf{H}_0(\mathsf{state})$$

$$\tilde{\mathsf{H}}_\infty(\mathcal{Z}_r|\mathsf{state}) \geq \mathsf{H}_\infty(\mathcal{Z}_r) - cvk = vk - cvk$$

$\mathcal{Z}_r$ is extracted by making no RO queries only with probability less than $2^{-vk(1-c)} < 2^{-k(1-c)}$ which is negligible for constant c (even in the worse case where the adversary replies with a single replica). Therefore, there is at least one RO query. Let $\mathcal{Q}_r$ be the indices in I that indicates the queries which are y-values of round r. That is, $\forall u \in \mathcal{Q}_r$, $\mathcal{A}_2$ queried T on $y_{r-1}^{(u)}$, and $T(y_{r-1}^{(u)}) = z_r^{(u)}$. Let $q_r = |\mathcal{Q}|$ denote the number of "relevant" r-round queries. For each query, either the state contains information about the response or not; we consider the two cases where the state stores $< B$ bits of information or $\geq B$ of information of a query response. If the state contains $< B$ bits of information about responses in round r, then $\mathcal{B}$ wins if the challenge is embedded in that response and we are done. Otherwise, the state contains information about each $z_r^{(u)}$ which means the state stores at least B bits of information for each query.

$$\mathsf{H}_\infty(\mathsf{state}) \geq q_r B$$

Now, let us consider the set of queries made with indices in $\mathcal{Q}_r$. For each $y_{r-1}^{(u)}, u \in \mathcal{Q}_r$, we can extract $z_{r-1}^{(u)}$ by computing the decoding. That is, $z_{r-1}^{(u)} = f_{pk_u}(y_{r-1}^{(u)})$, for $y_{r-1}^{(u)} \in \mathcal{D}$. These q_r elements are outputs of RO on different inputs, and therefore have full entropy. Let $\mathcal{Z}_{r-1} = z_{r-1}^{(u_1)}|| \cdots ||z_{r-1}^{(u_{q_r})}$ where each $u_i \in \mathcal{Q}_r$. We have $\mathsf{H}_\infty(\mathcal{Z}_{r-1}) = q_r k$. If $\mathcal{Z}_{r-1}$ can be extracted from $\mathcal{A}_2^T(\mathsf{state})$, either the state contains information about $z_{r-1}^{(i)}, \forall i \in \mathcal{Q}_r$, or $\mathcal{A}_2$ must make more RO queries. If there are no more queries, then $\mathsf{H}_\infty(\mathsf{state}) \geq vk$. Therefore, there must be more queries on inputs corresponding to the indices in $\mathcal{Q}_r$.

Let q_{r-1} be the number of relevant round $(r-1)$ queries. Define a set of query indices $\mathcal{Q}_{r-1}$, from which we can extract $\mathcal{Z}_{r-2} = z_{r-2}^{(u_1)}|| \cdots ||z_{r-2}^{(u_{q_{r-1}})}$, for $u_i \in \mathcal{Q}_{r-1}$. Again, for each query, either the state dedicates $< B$ bits of information, in which case $\mathcal{B}$ wins if the challenge is embedded in that response and we are done. Otherwise, the state stores at least B bits of information about each $y_{r-1}^{(u)}$, and therefore we have,

$$\mathsf{H}_\infty(\mathsf{state}) \geq q_r B + q_{r-1} B$$

Making a similar argument as before, there must be more RO queries corresponding to the indices in set $\mathcal{Q}_{r-1}$. Thus, we have, after r rounds, $\mathcal{Z}_r, \ldots, \mathcal{Z}_1$ are extracted from $\mathcal{A}_2(\mathsf{state})$, and we have

$$\mathsf{H}_\infty(\mathsf{state}) \geq \sum_{i=1}^{r} q_i B$$

Let $\mathcal{A}_2$ make RO queries in each round j for the replicas given by the indices in $\mathcal{Q}_j$ such that the reduction could not successfully embed a challenge in any of the responses. Then after r rounds, setting,

$$\sum_{i=1}^{r} q_i B - cvk = k$$

We get $\tilde{\mathsf{H}}_\infty(\mathcal{Z}_1|\mathsf{state}) \geq k$, when $r > k(cv+1)/B$.

Therefore, at some round $\ell \leq r$, the entropy of the response is full when making an RO query at round ℓ. That is, $\exists \ell \in [1,r], w \in [1,n]$ such that,

$$\tilde{\mathsf{H}}_\infty(z_\ell^{(w)}|\mathsf{state}) = k$$

When there are multiple blocks, we have $|\mathsf{state}| < cv\beta$. Since the permutation H is applied to the entire file concatenated with a random string r, before r rounds of T and f, the adversary can find files such that the output of H results in the same blocks only with probability $((qs)^2+1)/2^k$ where q is the number of queries and s the number of blocks. Then the above argument holds for each block independently.

The probability that the challenge is programmed into the RO response of one of the blocks of $z_\ell^{(w)}$ is the probability that $i^* = w, j^* = \ell$ which is $1/vrs > 1/nrs$ (in the worse case where $n = v$). Thus the probability that $\mathcal{B}$ wins is at least $\frac{\epsilon}{nrs} - 2^{-k(1-c)} - qs^2 2^{-k}$.

□

As any trapdoor function is also trivially B-leakage secure for $B = \log(k)$, we obtain the following corollary.

Corollary 1. *Assuming T, H are invertible random oracles, the construction* ReplicaEncoding = (rEnc, rDec) *using trapdoor permutation f is a secure replica encoding scheme for replication parameter n as per Definition 4. For number of rounds $r > \frac{(cn+1)k}{\log k}$, it is complete and c-sound with soundness error*

$$\epsilon \leq \left(\epsilon' + 2^{-k(1-c)} + qs^2 2^{-k}\right) nrs$$

where $k = \log|\mathcal{D}|$, $\mathcal{D}$ is the domain of f and T, s is the number of blocks, q the number of queries to the RO and the advantage of any adversary in inverting the permutation f is at most ϵ'.

Of course, for specific trapdoor permutations, it might be possible to assume B-leakage security for larger values of B thus achieving better round complexity.

4.2 From Replica Encodings to Proofs of Replication

We now construct a proof of replication scheme create, retrieve, $\mathcal{P}, \mathcal{V}$. The idea is very simple: to construct a proof of replication we use the replica encoding scheme from the previous section to create replicas, and then apply a proof of retrievability on the encoded replicas. The proof of security is also simple, as an adversary that breaks soundness for the proof of replication can be used to break the soundness property of the proof of retrievability scheme or the soundness of the replica encoding scheme.

The create procedure is formally described in Figure 6. The prover, and verifier algorithms $\mathcal{P}, \mathcal{V}$ are the same as the prover and verifier in the proof of retrievability. Finally, the retrieve algorithm simply runs the replica decoding algorithm rDec if the proof of retrievability accepts.

Let $\mathsf{PoR} = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ be a proof of retrievability scheme. Given a file $m \in \{0,1\}^{k'}$, and a replication factor n:

- For each $i \in [1, n]$
  - Run $R^{(i)} \leftarrow \mathsf{rEnc}(m, \kappa)$
  - $(\{\tilde{R}^{(i)}\}_i, \tau_i) = \mathsf{PoR.Gen}(R^{(i)})$
- Set $\mathsf{ver} = \{\tau_i\}_i$
- $\tilde{R}^{(i)}$ is sent to the server i for storage and ver is returned to the client.

Figure 6: $\mathsf{create}(m, n)$: Create replicated storage

Theorem 2. $\mathsf{PoRep} = (\mathsf{create}, \mathsf{retrieve}, \mathcal{P}, \mathcal{V})$ *is a proof of replication scheme for replication parameter n secure as per Definition 5. It is complete and c-sound with soundness error $\gamma \leq \delta + \epsilon$ where the underlying* PoR *scheme has soundness error δ, and the replica encoding scheme has soundness error ϵ.*

Proof. We first argue completeness: Given $R^{(i)}$ and pk, for encodings that are created honestly, an honest server can recover $m^* = \mathsf{retrieve}(R^{(i)})$. By completeness of the replica encoding scheme rEnc, we have $\mathsf{rDec}(R^{(i)}) = \mathsf{rDec}(y^{(i)}, pk^{(i)}) = m, \forall i$.

We now argue the soundness of the construction. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be an adversary, that wins the soundness game $\mathsf{sound}^{\mathcal{E}}_{\mathcal{A}_1,\mathcal{A}_2}$ with advantage γ. Let $(u, v) \leftarrow \mathsf{sound}^{\mathcal{E}}_{\mathcal{A}_1,\mathcal{A}_2}$.We consider the two cases:

Case 1. $u < v$. Let ext be the extractor of the PoR scheme, and let the file output by ext be $\{\tilde{R}^{(i)}\}_{i=1}^n$. By assumption that $u < v$ there must be an index $i \in [1, n]$ such that the adversary $\mathcal{A}_2$ succeeds in the audit protocol (i.e., $v_i = 1$), but $\tilde{R}^{(i)} \neq R^{(i)}$ (i.e., $u_i = 0$). By the soundness of the proof of retrievability scheme PoR, this happens only with probability δ.

Case 2. $|\mathsf{state}| \leq cv\beta$. In this case the adversary $\mathcal{A}_2^T$ succeeds in v audit protocols, and since $u \geq v$, the extractor $\mathcal{E}$ outputs $\tilde{R}^{(i)} = R^{(i)}$ for $i \in I \subset [1, n], |I| = v$. Let $(\mathcal{B}_1, \mathcal{B}_2)$ be an adversary whose task is to break the soundness of the replica encoding scheme rEnc. $\mathcal{B}_1$ interacts with $(\mathcal{A}_1, \mathcal{A}_2)$ in the soundness game. $\mathcal{B}_1$ receives a file $m \in \{0,1\}^{k'}$ from $\mathcal{A}_1$, and outputs m to its challenger. $\mathcal{B}_1$ receives n replica encodings $(R^{(1)}, \cdots, R^{(n)})$ from the challenger, where the bit length of each encoding is β. $\mathcal{B}_1$ runs the PoR on the replicas. $(\{\tilde{R}^{(i)}\}_i, \mathsf{ver}) \leftarrow \mathsf{PoR.Gen}\{R^{(i)}\}_i$ and returns $\{\tilde{R}^{(i)}\}_i$ to $\mathcal{A}_1$. $\mathcal{B}_1$ outputs as state whatever $\mathcal{A}_1$ outputs with $|\mathsf{state}| \leq cv\beta$. For every successful audit proof given by $\mathcal{A}_2$, $\mathcal{B}_2$ runs the extractor $\mathcal{E}(\mathsf{ver}, n, \kappa)$ of the scheme. Thus $\mathcal{B}_2$ outputs $\tilde{R}^{(i)} = R^{(i)}$ for each $i \in I$ with probability at least γ.

□

5 Dealing with Malicious Clients

We discuss here some limitations and possible extensions of our approach.

Our definition and construction so far has concentrated on the case where the client is honest. This is not a problem for our base use-case where a user wants to make sure they will be able to retrieve their files in the future, but it is a problem in the Filecoin use case where servers are rewarded for the files they store. In this case, we need to prevent against the so called *generation attack* and it is therefore important to have some security guarantees when the client is corrupt and might work with a set of corrupt servers to convince honest users that they store many replicas whereas in fact the replicas are generated "on-the-fly" for each proof.

Our solution from the previous section does not work in this case, as a corrupt user could share the trapdoor function secret key with the servers and now they can indeed encode a replica on the fly. If the client who owns the file is corrupt and is the only user involved in the encoding process, then the adversary knows everything about the encoding process, and a different solution is needed. One way to go is to involve several users in the encoding process and work under the assumption that at least one of them is honest.

In a Filecoin-like scenario one could implement such a solution by rewarding users for helping others to encode. We now describe two different approaches to such multi-user encoding.

Parallel Encoding. Given one-way trapdoor functions $f_1, ..., f_n$ that act on k-bit strings, we define a new function F on kn bit strings by $F(x_1, ..., x_n) = (f_1(x_1), ..., f_n(x_n))$, where each x_i is a k-bit string. It is clear that F is a one-way trapdoor function even with respect to an adversary who knows all but one of the trapdoors for $f_1, ..., f_n$. Namely, if the j'th trapdoor is unknown, we can take a challenge y_j, choose $y_i = f_i(x_i)$ for $i \neq j$ and random x_i and give $(y_1, ..., y_n)$ to the adversary. If he computes a preimage, then the j'th component is the answer to challenge y_j.

Note that our main result gives a construction that is secure based on any one-way trapdoor function and so it also works for F.

Now, in the practical use-case, we will assume that n user are involved, such that user i has f_i as part of his public key and knows the corresponding trapdoor. A public-key infrastructure is one of several ways to realize this. Then the n users can collaborate to encode: whenever we need to compute F, we can assume the input to the current round is known (initially it will be the file to encode), so each user i can apply the permutation oracle and compute and broadcast f_i^{-1} on his part of the result.

It follows immediately from the above that if at least one user is honest, then this construction results in a secure replica encoding. Note also that the contributions of each user can be verified by computing his function in the forward direction. Moreover, the overhead in encoding size and the cost per bit of encoding and decoding is the same as for the single honest user case. On the other hand, we need a number of rounds for the encoding protocol that equals the number of rounds in the encoding process.

Sequential Encoding. An obvious alternative is to do the encoding sequentially. Namely, the first user does an encoding of the input file using his (set of) trapdoor functions and broadcasts the result. The second user encodes the output of the first, etc. In the end, we have an encoding that is essentially done just like our original construction, only with more rounds. Note that one can decode the output of each user and check the result is correct.

The intuition behind the security of this approach is the following: under the assumption that at least one user is honest, we have the same security as in the original construction assuming that at least one of the users is honest. This is simply because the adversary does not know the trapdoors for the honest member, and his encoding process involves the same number of random oracle responses that we considered in the original proof. This approach increases the size of the encodings but not significantly (the complexity would grow from $O(|m| + \kappa)$ to $O(|m| + n \cdot \kappa)$ with n users). The cost of encoding and decoding in this solution is a factor n larger than for the single honest user case. On the other hand, the number of rounds to encode is n, which may be better than parallel encoding, depending on the concrete scenario.

6 Conclusions and Future Work

We gave two possible solutions to multi-user encoding above. However, there is also a solution of a different nature that comes to mind: namely we can share a trapdoor (say, an RSA key) between a set of users and have them collaborate to compute the encoding using that trapdoor function securely. This has the advantage that an encoding looks just like what an honest client would produce, we are not forced to have larger block size when many users are involved. Also, decoding is as fast as in the honest client case, and one can set up the protocol such that just one participating client needs to be honest in order for the secret key to not leak. On the other hand, encoding requires more work. For the encoding protocol, one can take advantage of the huge body of literature on distributed RSA key generation and distributed signing. Finding the optimal solution for this approach is left for future work.

We also leave as an open question the problem of finding a secure replica encoding where the number of rounds in the encoding process does not depend on the number of replicas.

References

[AAC+17] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 357–379. Springer, Heidelberg, December 2017.

[ABBK16] Frederik Armknecht, Ludovic Barman, Jens-Matthias Bohli, and Ghassan O Karame. Mirror: Enabling proofs of data replication and retrievability in the cloud. In *USENIX Security Symposium*, pages 1051–1068, 2016.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

[BH11] Ayad F Barsoum and M Anwar Hasan. On verifying dynamic multiple data copies over cloud servers. *IACR Cryptology ePrint Archive*, 2011:447, 2011.

[CFMJ18] Ethan Cecchetti, Ben Fisch, Ian Miers, and Ari Juels. Pies: Public incompressible encodings for decentralized storage. Cryptology ePrint Archive, Report 2018/684, 2018. `https://eprint.iacr.org/2018/684`.

[chi17] Chia network. https://chia.network/, 2017.

[CHK+16] Jean-Sébastien Coron, Thomas Holenstein, Robin Künzler, Jacques Patarin, Yannick Seurin, and Stefano Tessaro. How to build an ideal cipher: The indifferentiability of the Feistel construction. *Journal of Cryptology*, 29(1):61–114, January 2016.

[CKBA08] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. Mr-pdp: Multiple-replica provable data possession. In *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, pages 411–420. IEEE, 2008.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Heidelberg, August 2015.

[DKT16] Dana Dachman-Soled, Jonathan Katz, and Aishwarya Thiruvengadam. 10-round Feistel is indifferentiable from an ideal cipher. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 649–678. Springer, Heidelberg, May 2016.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing*, 38(1):97–139, 2008.

[DS16] Yuanxi Dai and John P. Steinberger. Indifferentiability of 8-round Feistel networks. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 95–120. Springer, Heidelberg, August 2016.

[DVW09] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 109–127. Springer, Heidelberg, March 2009.

[EK13] Mohammad Etemad and Alptekin Küpçü. Transparent, distributed, and replicated dynamic provable data possession. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 1–18. Springer, Heidelberg, June 2013.

[FBBG18] Ben Fisch, Joseph Bonneau, Juan Benet, and Nicola Greco. Proofs of replication using depth robust graphs, 2018. https://cyber.stanford.edu/bpase18.

[Fis18] Ben Fisch. Tight proofs of space and replication. Cryptology ePrint Archive, Report 2018/702, 2018. https://eprint.iacr.org/2018/702.

[HY10] Zhuo Hao and Nenghai Yu. A multiple-replica remote data possession checking protocol with public verifiability. In *Data, Privacy and E-Commerce (ISDPE), 2010 Second International Symposium on*, pages 84–89. IEEE, 2010.

[JK07] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 584–597. ACM Press, October 2007.

[Lab17a] Protocol Labs. Filecoin: A decentralized storage network. `https://filecoin.io/filecoin.pdf`, 2017.

[Lab17b] Protocol Labs. Proof of replication. `https://filecoin.io/proof-of-replication.pdf`, 2017.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, February 2004.

[Pie18] Krzysztof Pietrzak. Proofs of catalytic space. Cryptology ePrint Archive, Report 2018/194, 2018. `https://eprint.iacr.org/2018/194`.

[PPK+15] Sunoo Park, Krzysztof Pietrzak, Albert Kwon, Joël Alwen, Georg Fuchsbauer, and Peter Gaži. Spacemint: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528, 2015. `http://eprint.iacr.org/2015/528`.

[RD16] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 262–285. Springer, Heidelberg, October / November 2016.

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 90–107. Springer, Heidelberg, December 2008.

[VDJO+12] Marten Van Dijk, Ari Juels, Alina Oprea, Ronald L Rivest, Emil Stefanov, and Nikos Triandopoulos. Hourglass schemes: how to prove that cloud files are encrypted. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 265–280. ACM, 2012.