Proofs of Replicated Storage Without Timing Assumptions

Ivan Damgård, Chaya Ganesh, and Claudio Orlandi

{ivan,ganesh,orlandi}@cs.au.dk, 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 the 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 store a file m in the cloud or on some other decentralized network of servers. To make sure the file is accessible to both A and other users, 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.

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. 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 in 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. This 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. 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.

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 expensive.

In any case, the basic problem with all time-bounded schemes is that the encoding has to be made so slow that even a powerful server cannot encode faster than the time a proof takes. However, this slow-down also hurts the honest client every time he encodes.

We ask if we can do better in all the above aspects: can we have a proof of replication scheme that provably resists sybil and generation 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 requires a primitive we call a *hard encoding* which 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 a solution that works for a corrupt client, at the expense of using more communication.

Very roughly speaking, the idea 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. Thus, while we do not prove that the adversary must store the concatenation of all replicas, we do ensure that, in terms of storage cost, he has no incentive to do anything else. Significantly, compared with previous proposed solutions to the problem, 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. This clearly makes implementing proof of replications much easier, since one does not need to worry about finding an appropriate value for the verifier timeout.

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 Armknecht et al. [ABBK16] 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 latter protocol, in addition, uses RSA time-lock puzzles which results in a protocol with a time-bounded property that we elaborate on below.

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] 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 generation 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 infeasible. 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, 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 completely at least one replica, say the i'th one. 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: since the adversary now has no information on $f^{-1}(T(m||i))$, he also has no information on what T outputs on input $m||i$ (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 challenge value for the one-way permutation in this position by an argument similar to the above.

To highlight exactly which properties of the one-way permutation we use, we introduce a notion we call a hard encoding. A hard encoding scheme $\mathsf{hardEnc}$ consists of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}(1^\kappa)$ outputs an encoding and a decoding key; Enc takes a message $m \in \{0,1\}^*$ and the encoding key, and outputs an encoding, $c \leftarrow \mathsf{Enc}(m, \mathsf{ek})$; Dec takes an encoding, the decoding key and returns a message. Of course, encoding followed by decoding should return the correct message, but otherwise the main assumption is that computing the encoding of a given message should be hard unless you are given the encoding key. On the other hand, anyone can decode (in practice, the decoding key would be a part of the coded message).

This can trivially be instantiated from a trapdoor 1-way permutation by using the trapdoor as the encoding key. Note, however, that we actually do not need the full power of trapdoor 1-way permutations: in our application, we encode once and after this we only need decoding. So the encoding key can be deleted after use. In fact, we can go a step further: if we only want to encode a single message, we can collapse the Gen and Enc algorithms to a single one that takes the message m as input, and outputs a decoding key dk and an encoding c such that $\mathsf{Dec}(c, \mathsf{dk}) = m$. To instantiate this we just need, very informally speaking, a kind of "punctured" function f that is one-way, but can be generated based on a given value m, such that one learns x with $f(x) = m$. This trivially generalizes to encoding of more than one message at once. We do not formalize this line of reasoning in this paper, nor do we know any instantiations other than 1-way permutation. We just mention this as a hint that our scheme might be instantiable from a weaker assumption and leave this as an interesting open problem.

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 $y \overset{R}{\leftarrow} \mathcal{Y}$ to mean sampling a value y uniformly from the set $\mathcal{Y}$.

2.1 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 $f_{sk}^{-1}(y) = y^d \bmod N$

Define the advantage of A as

$$\mathsf{Adv}_{\mathcal{A}}^{rsa} = \Pr[f_{pk}(z) = y : (pk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa), x \overset{R}{\leftarrow} \mathbb{Z}_N^*, z \leftarrow A(pk, f_{pk}(x))]$$

Definition 1. *The RSA inversion problem is said to be hard if for any $\mathcal{A}$ running in time polynomial in κ, the $\mathsf{Adv}_{\mathcal{A}}^{rsa}$ is negligible; there exists a negligible function* negl *such that*

$$\mathsf{Adv}_{\mathcal{A}}^{rsa} \leq \mathsf{negl}(\kappa)$$

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.

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

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 2 (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.

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.

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 Hard Encoding

We introduce the notion of a hard encoding which is useful in the context of replicated storage. A hard encoding encodes a message such that the decoding procedure is easy to compute for everyone, but the encoding procedure is only easy if one has access to the encoding key. Hard encodings can be realized using trapdoor permutations, where the secret key of the permutation is erased after computing the encoding.

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 $(y^{(1)}, \cdots, y^{(n)})$ 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 $v = \sum_{i=1}^{n} v_i$.

A hard encoding scheme $\mathsf{hardEnc}$ consists of algorithms $(\mathsf{Enc}, \mathsf{Dec})$, where $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}(1^\kappa)$ outputs an encoding and a decoding key; Enc takes a message $m \in \{0,1\}^*$ and the encoding key, and outputs an encoding, $c \leftarrow \mathsf{Enc}(m, \mathsf{ek})$; Dec takes an encoding, the decoding key and returns a message. For the sake of concreteness, throughout the paper, we use the RSA trapdoor permutation to implement hard encodings.

- Gen: $((e,N), d) \leftarrow \mathsf{KeyGen}(1^\kappa)$ Set $\mathsf{ek} = d, \mathsf{dk} = (e, N)$
- Enc: For a message $m \in \mathbb{Z}_N^*$, given ek, compute $c = m^d \bmod N$
- Dec: Given $c \in \mathbb{Z}_N^*$, and dk, output $m = c^e \bmod N$

3.2 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 3 (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. 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.*

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

3.3 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 $y^{(1)}, \cdots, y^{(n)}$ together with verification information ver. The replicas $y^{(i)}$ are sent to the server(s) 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 the server 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 $\langle \mathcal{P}_i(\tilde{y}^{(i)}), \mathcal{V}(\mathsf{ver}, i)\rangle$. Note that each server should be able to prove to the verifier that they are storing the file independently and that, when considering soundness, we assume all provers are under the control of a monolithic adversary. We require the scheme to satisfy completeness and soundness properties. We give a formal definition below. All algorithms are parametrized by a security parameter, which we omit in our description below.

Definition 4 (Proof of Replication). *A scheme* $\mathsf{PoRep} = (\mathsf{create}, \mathsf{retrieve}, \mathcal{P}, \mathcal{V})$ *where,*

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

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

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

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

- *Completeness. For an honest client and honest server,*
  - *for* $(y^{(1)}, \cdots, y^{(n)}, \mathsf{ver}) \leftarrow \mathsf{create}(m, n), m_i^* = \mathsf{retrieve}(y^{(i)}), m_i^* = m \ \forall i \in [1, n]$
  - *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. 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.*
  $$\Pr\left[u < v \vee |\mathsf{state}| < cvk | (u, v) \leftarrow \mathsf{sound}^{\mathcal{E}}_{\mathcal{A}_1,\mathcal{A}_2}\right] \leq \mathsf{negl}(\kappa)$$

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. But a malicious server could forget bits (a constant fraction) of each encoding, instead of completely forgetting any one replica. Now, to pass the audit, the server has to compute a preimage of the underlying trapdoor permutation, but *given* a constant fraction of bits of the preimage. We now apply a random permutation on the message concatenated with a short seed per replica, prior to using the hard

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 $(y^{(1)}, \cdots, y^{(n)}, \mathsf{ver}) \leftarrow \mathsf{create}(m, n)$ and returns $(y^{(1)}, \cdots, y^{(n)})$ to $\mathcal{A}_1$.
- $\mathcal{A}_1$ outputs a state $\mathsf{state} \leftarrow \mathcal{A}_1(y^{(1)}, \cdots, y^{(n)})$
- 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.

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

- 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).

encoding. Again, the server can simply store few bits of the encoding along with a short seed and pass the audit. The final idea behind our construction is to iterate the permuation plus encoding as a round function sufficiently many times. Intuitively, now preimage information is to be stored at each round, and the total exceeds the bound necessary for replicated storage after a sufficient number of rounds. 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 vector 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".

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 2 and 3.

Theorem 1. *Assuming T is an invertible random oracle, the construction* $\mathsf{ReplicaEncoding} = (\mathsf{rEnc}, \mathsf{rDec})$ *is a secure replica encoding scheme for message length k', replication parameter n as per Definition 3. For number of rounds $r > cn$, it is complete and c-sound with soundness error $\epsilon \leq (\epsilon' + 2^{-nk(1-c)})cn^2$ where $k = k' + \kappa$, the advantage of any adversary in inverting the RSA permutation as per Definition 1 is at most ϵ'.*

Proof. Completeness. For n encodings that are created honestly, $R^{(i)} \leftarrow \mathsf{rEnc}(m, \kappa)$, $m^* = \mathsf{rDec}(R^{(i)})$. Since T is a permutation and the hard encoding is the RSA permutation, $m^* = m, \forall i$.

Soundness. Assume there exists an adversary (A_1, A_2) such that $\Pr\left[|\mathsf{state}| < cvk | v \leftarrow \mathsf{sound}_{\mathcal{A}_1,\mathcal{A}_2}\right] > \epsilon$. Therefore, the adversary 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 A_2 outputs correctly. We argue that if the state is too small, it 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, (and queried to T), which we use to invert the trapdoor permutation of the hard

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

- Choose a string r^i uniformly at random from $\{0,1\}^\kappa$, and let $y_0^{(i)} = m||\mathsf{r}^i$. Let $k = k' + \kappa$.
- Let $\mathsf{hardEnc} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a hard encoding scheme. $(\mathsf{ek}^{(i)}, \mathsf{dk}^{(i)}) = \mathsf{hardEnc.Gen}(\kappa)$. Divide $y_0^{(i)}$ into s blocks such that each block is in $\mathbb{Z}_N$, where $\mathsf{dk}^{(i)} = (e, N)$. Let $T_N : \mathcal{D} \to \mathcal{D}$ be viewed as an invertible RO, where the domain and range $\mathcal{D}$ is $(\mathbb{Z}_N)^s$. Iterate the following round function: For each round j from 1 to r, define
  - Apply the RO T_N,
  $$z_j = T_N(y_{j-1}^{(i)})$$
  - Parse z_j as $Z_{1j}||\cdots||Z_{sj}$ where each $Z_{tj} \in \mathbb{Z}_N$. Apply the hard encoding block-wise. For each $t \in [1, s]$
  $$Y_{tj}^{(i)} = \mathsf{hardEnc.Enc}(Z_{tj}, \mathsf{ek}^{(i)})$$
  - Let $y_j^{(i)} = Y_{1j}^{(i)}||\cdots||Y_{sj}^{(i)}$
- Let $R^{(i)} = (y^{(i)}, \mathsf{dk}^{(i)})$, where $y^{(i)} = y_r^{(i)}$
- Return $R^{(i)}$

Figure 2: The Replica Encoding Algorithm $\mathsf{rEnc}(\kappa, m)$

For a replica $R^{(i)} = (y^{(i)}, \mathsf{dk}^{(i)})$, let $y_r = y^{(i)}, (e, N) = \mathsf{dk} = \mathsf{dk}^{(i)}$. For each round j from r down to 1, compute

- Round j:
  - Parse $y_j^{(i)}$ as $Y_{1j}^{(i)}||\cdots||Y_{sj}^{(i)}$. Decode the hard encoding block-wise, for each $t \in [1, s]$
  $$Z_{tj} = \mathsf{hardEnc.Dec}(Y_{tj}^{(i)}, \mathsf{dk})$$
  - Let $z_j = Z_{1j}||\cdots||Z_{sj}$
  $$y_{j-1}^{(i)} = T_N^{-1}(z_j)$$
- Parse $y_0 = m||\mathsf{r}$ where m is the first k' bits of y_0. Return m.

Figure 3: The replica decoding algorithm $\mathsf{rDec}(R^{(i)})$

encoding. The high level idea is that since the state is small, in round r, A_2 must have learned some of the z values of round r from responses of T. Therefore, A_2 must make the correct queries, which are the y values of round $r-1$. From these queries, we can extract the z values of round $r-1$ by simply decoding, and we make a similar argument on these z values. 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, until we hit a round where the response of T for one of A_2's queries must have full entropy from A_2's point of view. We use this response 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 of the underlying $\mathsf{hardEnc}$ scheme. $\mathcal{B}$ receives $(\hat{e}, \hat{N})$, a challenge $\hat{x}$, and wins if it outputs $\hat{y}$ such that $\hat{y}^{\hat{e}} = \hat{x} \bmod \hat{N}$. $\mathcal{B}$ interacts with (A_1, A_2) in the soundness game. $\mathcal{B}$ receives a file $m \in \{0,1\}^{k'}$ from A_1. $\mathcal{B}$ creates encoded replicas honestly, except for the following. It chooses a random $i^* \in [1,n]$, and creates replica i^* in the following way. It chooses a random value y and computes the trapdoor permutation in the forward direction with the challenge key $(\hat{e}, \hat{N})$ for each block in y to obtain z. It then programs the oracle T such that $T(m) = z$. This is repeated for each round $j \in [1,r]$. That is, $\mathcal{B}$ defines the i^*th replica in rEnc as follows.

- For $i = i^*$, let $y_0^{(i)}$ be a uniformly random string of length $k = k' + \kappa$. Let r^i be a uniformly random string of length κ. Parse $y_0^{(i)}$ as $Y_{10}^{(i)}||\cdots||Y_{s0}^{(i)}$, $Y_{t0}^{(i)} \in Z_{\hat{N}}$. For each $t \in [1,s]$
$$Z_{t0} = (Y_{t0}^{(i)})^{\hat{e}} \bmod \hat{N}$$
- Let $z_0 = Z_{10}||\cdots||Z_{s0}$
- Define
$$T_{\hat{N}}(m||\mathsf{r}^{(i)}) = z_0$$
In each round j from 1 to r, define
  - Choose random $y_j^{(i)} \in \{0,1\}^k$. Parse $y_j^{(i)}$ as $Y_{1j}^{(i)}||\cdots||Y_{sj}^{(i)}$. For each $t \in [1,s]$
$$Z_{tj} = (Y_{tj}^{(i)})^{\hat{e}} \bmod \hat{N}$$
  - Let $z_j = Z_{1j}||\cdots||Z_{sj}$
  - Define
$$y_{j-1}^{(i)} = T_{\hat{N}}^{-1}(z_j)$$
- Let $y^{(i^*)} = y_r^{(i)}$ and $\mathsf{dk}^{(i^*)} = (\hat{e}, \hat{N})$
- Return $R^{(i^*)} = (y^{(i^*)}, \mathsf{dk}^{(i^*)})$

$\mathcal{B}$ responds to any other oracle queries of A_1 honestly, and finally gives $(R^{(1)}, \cdots, R^{(n)})$ to A_1, where $R^{(i)}$ for $i \neq i^*$ is created honestly. A_1 outputs a state state. Now, $\mathcal{B}$ interacts with A_2. It runs A_2 on state, and receives and responds to A_2's oracle queries in the following way. $\mathcal{B}$ randomly chooses $j^* \in [1,r]$. If A_2 queries T on $y_{j^*-1}^{(i^*)}$, $\mathcal{B}$ chooses a random block $t^* \in [1,s]$, sets the response to embed its challenge $\hat{x}$ in the following way. Set $Z_{t^*j^*} = \hat{x}$, and choose Z_{tj^*} for all $t \neq t^*$ uniformly, and for $z_j^* = Z_{1j^*}||\cdots||Z_{sj^*}$,
$$T_{\hat{N}}(y_{j^*-1}^{(i^*)}) = z_j^*$$

The rest of the queries are answered honestly. If A_2 makes a query $y_{j^*}^{(i^*)} = Y_{1j^*}^{(i^*)}||\cdots||Y_{sj^*}^{(i^*)}$ with $Y_{tj^*}^{(i^*)} = \hat{y}$ such that, $\hat{y}^{\hat{e}} = \hat{x} \bmod \hat{N}$, $\mathcal{B}$ outputs $\hat{y}$. If there is no such query, $\mathcal{B}$ outputs $\perp$. We now argue that $\mathcal{B}$ wins with probability at least $\frac{\epsilon}{cn^2} - 2^{-nk(1-c)}$. We have $|\mathsf{state}| < cvk, c < 1$. Consider the min-entropy of the random variable state, which is at most the bit length, $\mathsf{H}_\infty(\mathsf{state}) \leq cvk$. A_2 on state returns, for some $I \subset [1,n], |I| = v$,
$$\{R^{(i)}\}_{i \in I} = A_2(\kappa, \mathsf{state})$$

for $R^{(i)} = (y_r^{(i)}, \mathsf{dk}^i)$. Let $\mathsf{dk}_i = (e_i, N_i)$. Each $y_r^{(i)}$ is parsed as $Y_{1r}||\cdots||Y_{sr}$, for $Y_{tr} \in Z_{N_i}$ and can be decoded block-wise to obtain $z_r^{(i)} = Z_{1r}||\cdots||Z_{sr}$ such that, $Z_{tr} = \mathsf{hardEnc.Dec}(Y_{tr}, \mathsf{dk}^i)$ for each $t \in [1, s]$.

Let $\mathcal{Y}_r = y_r^{(i_1)}||\cdots||y_r^{(i_v)}$ and $\mathcal{Z}_r = z_r^{(i_1)}||\cdots||z_r^{(i_v)}, i_j \in I$. Each $z_r^{(i)}$ is the output of RO, and is therefore 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, $\mathsf{H}_\infty(\mathcal{Z}_r) = nk$. Since $\mathcal{Z}_r$ can be extracted from $A_2^T(\mathsf{state})$, either the state contains information about each $z^{(i)}$ in $z^{(1)}||\cdots||z^{(n)}$, or 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),

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

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

$\mathcal{Z}_r$ is extracted by making no RO queries only with probability $< 2^{-nk(1-c)}$. 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, $\forall u \in \mathcal{Q}_r$, 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.

Let $\mathcal{S} = \{1, \cdots, n\} \backslash \mathcal{Q}_r, |\mathcal{S}| = t_r = n - q_r$. Since t_r blocks of $z_r^{(i)}$, $i \in [1, n]$ were extracted from A_2 without making a corresponding query, that is, A_2 did not query T on $y_{r-1}^{(u)}$, for $u \in \mathcal{S}$ such that $T(y_{r-1}^{(u)}) = z_r^{(u)}$. Therefore,

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

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 block-wise. That is, $z_{r-1}^{(u)} = Z_{1(r-1)}||\cdots||Z_{s(r-1)}$ such that, $Z_{t(r-1)} = \mathsf{hardEnc.Dec}(Y_{t(r-1)}, \mathsf{dk}^u)$ for each $t \in [1, s]$, where $y_{r-1}^{(u)} = Y_{1(r-1)}||\cdots||Y_{s(r-1)}$, $Y_{tr} \in Z_{N_u}$.

These q_r elements are outputs of RO, 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 $A_2^T(\mathsf{state})$, either the state contains information about $z_{r-1}^{(i)}, \forall i \in \mathcal{Q}_r$, or A_2 must make more RO queries. We have,

$$\tilde{\mathsf{H}}_\infty(\mathcal{Z}_{r-1}|\mathsf{state}) \geq q_r k - cnk$$

If there are no more queries,

$$\mathsf{H}_\infty(\mathsf{state}) \geq kt_r + kq_r$$

Since $t_r + q_r = n$ and $\mathsf{H}_\infty(\mathsf{state}) \leq cnk$, there must be more queries. 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 $(r-1)$-round 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 $u_i \in \mathcal{Q}_{r-1}$. We know $\mathsf{H}_\infty(\mathcal{Z}_{r-2}) = q_{r-1}k$. Let $\mathcal{Z}^j = \mathcal{Z}_j||\mathcal{Z}_{j+1}||\cdots||\mathcal{Z}_{r-1}$

$$\tilde{\mathsf{H}}_\infty(\mathcal{Z}^{r-2}|\mathsf{state}) \geq q_{r-1}k + q_r k - cnk$$

$$\mathsf{H}_\infty(\mathsf{state}) \geq kt_r + kt_{r-1} + kq_{r-1}$$

If there are no more queries, $q_{r-1} + t_{r-1} = q_r$; Therefore, again, there must more RO queries corresponding to the indices in set $\mathcal{Q}_{r-1}$. Thus, we have, after r rounds, $\mathcal{Z}_r, \cdots, \mathcal{Z}_1$ are extracted from $A_2(\mathsf{state})$, and we have

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

A_2 makes RO queries in each round j for the replicas given by the indices in $\mathcal{Q}_j$. After r rounds, for $\mathcal{Z}^1 = \mathcal{Z}_1||\mathcal{Z}_2||\cdots||\mathcal{Z}_{r-1}$,

$$\tilde{\mathsf{H}}_\infty(\mathcal{Z}^1|\mathsf{state}) \geq \sum_{i=1}^{r} q_i k - cnk$$

Setting,

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

Since there is at least one query in each round, we get $\tilde{\mathsf{H}}_\infty(\mathcal{Z}^1|\mathsf{state}) \geq k$, when $r > cn$. 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,

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

Thus $z_\ell^{(w)} = T(y_{\ell-1}^{(w)})$ with $Z_{t^* j^* - 1} = \hat{x}$ is the programmed RO response, is the probability that $i^* = w, j^* = \ell$ which is $1/nr$. Thus the probability that $\mathcal{B}$ wins is at least $\frac{\epsilon}{cn^2} - 2^{-nk(1-c)}$.

□

On instantiating the oracle T. As stated, each encoding key (e, N) defines a different random oracle T. We require that T be indifferentiable from a random permutation. The indifferentiablity framework, first proposed by Maurer et al [MRH04], informally says that given ideal primitives G and H, a construction C^G is indifferentiable from H, if there exists a simulator S with oracle access to H such that (C^G, G) is indistinguishable from (H, S^H). Coron et al [CHK⁺16] showed that a 14-round Fiestel 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.

Note that the input and output of our random oracles T are vectors of elements in $\mathbb{Z}_N$, where N is part of the public decoding key. 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 $H : (\mathbb{Z}_N)^{\ell/2} \to (\mathbb{Z}_N)^{\ell/2}$ we can define $\mathcal{F} : (\mathbb{Z}_N)^\ell \to (\mathbb{Z}_N)^\ell$ as follows:

$$\mathcal{F}^H(\vec{x}) = \vec{s}||\vec{t}, \text{ where } \vec{s} = \vec{x} \circ H(\vec{r}) \bmod N, \vec{t} = \vec{r} \circ H(\vec{s}) \bmod N$$

where $\circ$ is the Hadamard product. Note that $\mathcal{F}$ is invertible except with negligible probability i.e., if $\mathcal{F}$ is not invertible then a non-trivial factor of N is found.

4.2 From Replica Encodings to Proofs of Replication

We now construct a proof of replication scheme $\mathsf{create}, \mathsf{retrieve}, \mathcal{P}, \mathcal{V}$. 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 4. The prover, and verifier algorithms $\mathcal{P}, \mathcal{V}$ are the same as the prover and verifier in the proof of retrievability. Finally, the $\mathsf{retrieve}$ algorithm simply runs the replica decoding algorithm rDec if the proof of retrievability accepts.

Theorem 2. $\mathsf{PoRep} = (\mathsf{create}, \mathsf{retrieve}, \mathcal{P}, \mathcal{V})$ *is a proof of replication scheme for message length k' and replication parameter n secure as per Definition 4. 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 for message length k'.*

Proof. We first argue completeness: Given $R^{(i)}$ and dk, 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 $\mathsf{rDec}(R^{(i)}) = \mathsf{rDec}(y^{(i)}, \mathsf{dk}^{(i)}) = m$, $\forall i$.

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

Figure 4: $\mathsf{create}(m, n)$: Create replicated storage

We now argue the soundness of the construction. Let (A_1, 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 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. $|\mathsf{state}| \leq cvk$. In this case the adversary 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 (A_1, A_2) in the soundness game. $\mathcal{B}_1$ receives a file $m \in \{0,1\}^k$ from A_1, and outputs m to its challenger. $\mathcal{B}_1$ receives n replica encodings $(R^{(1)}, \cdots, R^{(n)})$ from the challenger $\mathcal{B}_1$ 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 A_1. $\mathcal{B}_1$ outputs as state whatever A_1 outputs with $|\mathsf{state}| \leq cvk$. For every successful audit proof given by 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 Removing Trust in the Client

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 RSA 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.

We identify two approaches for dealing with this problem which we sketch here:

1. The first is to re-introduce timing assumptions (as in previous constructions) and instantiate the trapdoor in our construction using RSA with small exponent (i.e., $e = 3$). Now decoding would be much faster then encoding, even if the adversary knew the secret key (in this case the adversary can optimize the encoding procedure using the Chinese-Remainder Theorem).

2. The second approach is to have multiple users encode the files in a sequential way. Note that, since the files are publicly retrievable, any user can add a layer of encoding. In the end, we have an encoding that is essentially done just like our original construction, only with more rounds. Note that it is still possible to decode and check the result is correct and that this does not significantly increase the size of the encodings (the complexity would grow from $O(|m| + \kappa)$ to $O(|m| + n \cdot \kappa)$ with n users). As an implementation detail, we note that one needs to deal with the fact that different users have different public keys and therefore RSA moduli. This can be done by maintaining the same number of blocks between the different encodings (and ensuring that the moduli of the users are increasing, so that the block-size of user i fits into a block of user $i + 1$), or by parsing the encoding of user i, which is a word of elements between 1 and N_i, into a word of elements between 1 and N_{i+1} (at the price of adding few extra blocks if necessary).

   By a straightforward extension of the proof from the previous section, we can show that under the assumption that at least one user is honest, we have the same security as in the original construction. This is simply because the adversary does not know the secret RSA key for the honest member, and his encoding process involves the same number of random oracle responses that we considered in the original proof.

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. Cryptology ePrint Archive, Report 2018/601, 2018. https://eprint.iacr.org/2018/601.

[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.

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

[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.