

# Proofs of Catalytic Space

Krzysztof Pietrzak<sup>1</sup>

Institute of Science and Technology Austria, Austria  
pietrzak@ist.ac.at

---

## Abstract

---

Proofs of space (PoS) [Dziembowski et al., CRYPTO'15] are proof systems where a prover can convince a verifier that he “wastes” disk space. PoS were introduced as a more ecological and economical replacement for proofs of work which are currently used to secure blockchains like Bitcoin. In this work we investigate extensions of PoS which allow the prover to embed useful data into the dedicated space, which later can be recovered.

Our first contribution is a **security proof** for the original PoS from CRYPTO'15 in the random oracle model (the original proof only applied to a restricted class of adversaries which can store a subset of the data an honest prover would store). When this PoS is instantiated with recent constructions of maximally depth robust graphs, our proof implies basically optimal security.

As a second contribution we show three different extensions of this PoS where useful data can be embedded into the space required by the prover. Our security proof for the PoS extends (non-trivially) to these constructions. We discuss how some of these variants can be used as **proofs of catalytic space** (PoCS), a notion we put forward in this work, and which basically is a PoS where most of the space required by the prover can be used to backup useful data. Finally we discuss how one of the extensions is a candidate construction for a **proof of replication** (PoR), a proof system recently suggested in the Filecoin whitepaper.

**2012 ACM Subject Classification** Theory of computation → Interactive proof systems

**Keywords and phrases** Proofs of Space, Proofs of Replication, Blockchains

**Digital Object Identifier** 10.4230/LIPIcs.ITCS.2019.59

**Related Version** A full version of the paper is available at [29], <https://eprint.iacr.org/2018/194.pdf>.

## 1 Introduction

### 1.1 Proofs of Space (PoS)

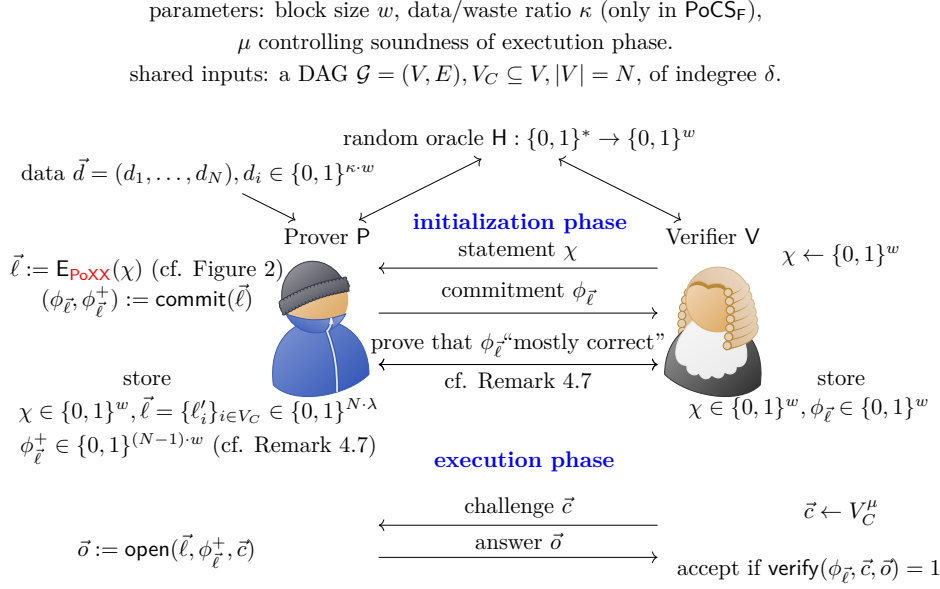
A proof of space (PoS) [16, 30, 3] is an interactive proof system in which a prover  $P$  can convince a verifier  $V$  that it “wastes” a large amount of disk-space. PoS were suggested as an alternative to proofs of work (PoW), which are currently used for securing blockchains including Bitcoin and Ethereum. PoS-based proposals include Spacemint [27] and the Chia network [1]. In the full version of this paper [29] we provide more discussion on sustainable blockchains and PoS.

The core of the pebbling-based PoS [16, 30] is a mode of operation for hash-functions  $E_{\text{PoS}}$ , which is specified by a directed acyclic graph (DAG)  $\mathcal{G} = (V, E)$  with a dedicated set  $V_C \subseteq V$  of  $|V_C| = N$  “challenge nodes”. The constructions in [16, 30] mostly differ in

---

<sup>1</sup> This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 682815/TOCNeT).





■ **Figure 1** Illustration of protocol structure of the proof of space  $\text{PoS}_o$ , our proofs of catalytic space  $\text{PoCS}_F, \text{PoCS}_\phi$  and the proof of replication  $\text{PoR}$  (replace  $\text{PoXX}$  in the figure with any of those). In  $\text{PoCS}_F$   $\kappa$  is a parameter, in  $\text{PoCS}_\phi, \text{PoR}$  set  $\kappa = 1$  and for  $\text{PoS}_o$  set  $\kappa = 0$  (i.e.,  $\vec{d}$  is empty) in the figure. The label size  $\lambda$  is  $w(\kappa + 1)$  in  $\text{PoCS}_F$  and  $w$  in  $\text{PoS}_o, \text{PoR}$  and  $\text{PoCS}_\phi$ .

what type of graphs are used. The only input  $E_{\text{PoS}_o}$  takes is a short statement  $\chi$  which is used to sample a hash function  $H_\chi$  (modelled as random oracle in all our proofs), and it outputs a large file  $\vec{\ell} = \{\ell_i\}_{i \in V_C}$  which P must store. P sends a commitment  $\phi_{\vec{\ell}}$  to  $\vec{\ell}$  to V. To check the prover really stores this file, the verifier can occasionally send a random challenge  $i \in V_C$  to the prover, who then must open the label  $\ell_i \in \vec{\ell}$  of this file. If such audits happen sufficiently often, the rational thing for P to do is to store  $\vec{\ell}$ , and not recompute labels as they are requested. The high level proof structure of this PoS, denoted  $\text{PoS}_o$ , is illustrated in Figure 1, the underlying mode of operation, denoted  $E_{\text{PoS}_o}$ , is illustrated in Figure 2.

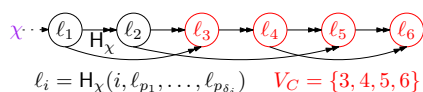
## 1.2 An Unconditional Security Proof for the [16] PoS

Informally, the security we want from a PoS is as follows: if a malicious prover  $\tilde{P}$  dedicates slightly less space than the honest prover would after the initialization phase, say  $(1 - \epsilon) \cdot N$  instead  $N$  for some small  $\epsilon > 0$ , then it should be “expensive” for him to pass the audit. Note that  $\tilde{P}$  can always pass the audit by simply recomputing the entire  $\vec{\ell}$  right before the audit, so the best we can hope for is that passing the audit is almost as expensive for  $\tilde{P}$  as it is to compute the entire  $\vec{\ell}$ .

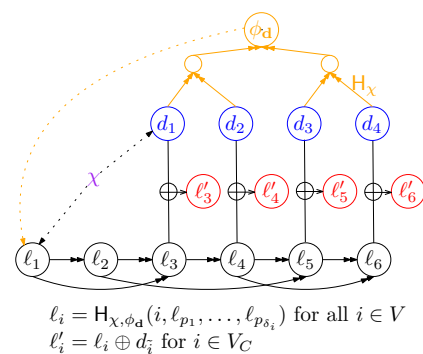
The first contribution in this paper is a security proof that shows  $\text{PoS}_o$  is a secure PoS in the random oracle model (Corollary 8 in §7.2). The existing proof from [16] only showed security against restricted adversaries who store a subset of the data  $\vec{\ell}$  an honest prover would store, but didn’t imply anything against more general adversaries who can store an arbitrary function of this data. We discuss this in more detail in §5.

When we instantiate  $E_{\text{PoS}_o}$  with recent constructions of depth-robust graphs, the security we get is basically optimal. Informally, for any  $\epsilon > 0$ , we can chose parameters such that any cheating prover who dedicates only an  $1 - \alpha$  fraction of the required space will fail to *efficiently* answer an  $\alpha - \epsilon$  fraction of the challenges (which simply ask to open some

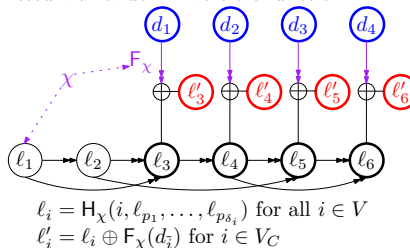
$E_{\text{PoS}_\circ}$ : The **proof of space** from [DFKP15] instantiated with (a toy example of) a depth-robust graph.



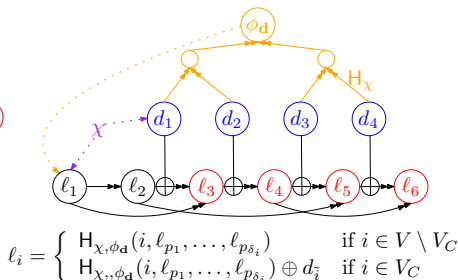
$E_{\text{PoCS}_\phi}$ : Our **proof of catalytic space** where the data is committed by a standard Merkle tree commitment  $\phi_d$ . The hash function for this commitment depends on  $\chi$ , the hash function for the labelling also on  $\phi_d$ .



$E_{\text{PoCS}_F}$ : Our **efficiently updatable proof of catalytic space** where the catalytic data committed via random invertible function  $F$ .



$E_{\text{PoR}}$ : Our **proof of replication** is similar to  $E_{\text{PoCS}_\phi}$ , but the data is XOR'ed to the labels as the computation goes on, not just at the end.



■ **Figure 2** Illustration of the graph based modes of operation used in the proof of space  $\text{PoS}_\circ$ , proof of catalytic space  $\text{PoCS}_\phi$  and its efficiently updatable variant  $\text{PoCS}_F$  and the proof of replication  $\text{PoR}$ . We use a toy example of a depth robust DAG  $\mathcal{G} = (V, E)$ ,  $V = \{1, \dots, 6\}$  with  $V_C = \{3, \dots, 6\}$  being the challenge nodes. The embedded data is shown in blue, the labels the prover stores are in red. The values represented by all nodes are in  $\{0, 1\}^w$ , except the bold nodes in  $\text{PoCS}_F$ , where the  $d_i$  are in  $\{0, 1\}^{w \cdot \kappa}$  and the  $l_i, l'_i, i \in V_C$  are in  $\{0, 1\}^{w \cdot (\kappa+1)}$ .

blocks in the file  $\vec{\ell}$  the prover is committed to). Thus, if say  $\alpha = 2\epsilon$ , the prover fails on an  $\epsilon$  fraction, and we can amplify this to be overwhelmingly close to 1 by using  $O(1/\epsilon)$  challenges in parallel. Above, with “efficiently recover”, we mean it needs parallel time<sup>2</sup>  $N$ , this is basically optimal in terms of time complexity, as running the entire initialization phase takes only (sequential) time  $4N$ . We will discuss how our proof compares with existing results in more detail in §2.

The efficiency of our schemes (i.e., proof size, proof generation time, proof verification time) are all in  $O(\log N)$ , where the hidden constant depends on the above mentioned  $\epsilon$  (i.e., the constant grows as  $\epsilon$  goes to 0).

### 1.3 Embedding Useful Data into a PoS

The file  $\vec{\ell} := E_{\text{PoS}_\circ}(\chi)$  the prover is supposed to store just wastes disk space, and cannot be used for anything useful. This makes sense, as after all a PoS is supposed to prove the dedicated space is wasted.

In this paper we investigate the setting where the space dedicated towards the PoS can at the same time be used to encode some useful data  $\vec{d}$ . We identify two applications for such objects, *proofs of replication* (PoR), which were (informally) introduced in the Filecoin

<sup>2</sup> Our proof is in the random oracle model, and parallel time  $N$  means  $N$  rounds of queries, where in each round one can make many queries in parallel. In sequential time just one query is allowed.

paper [23] and *proofs of catalytic space* (PoCS), which we introduce and motivate in this work. The naming of the latter is inspired by *catalytic space computations* [9, 10], which are computations that can be done in small space, but only if one is additionally given “catalytic space”. This space is initially filled with arbitrary (potentially incompressible) data, and must be in the same state after the computation finishes. It thus functions like a catalyst in chemical reactions.

We introduce three new proof systems which allow for such embedded data. Two of them are intended to be used as PoCS, denoted  $\text{PoCS}_\phi$  and  $\text{PoCS}_F$ . The  $\text{PoCS}_F$  scheme has a worse *rate* than  $\text{PoCS}_\phi$ , by which we mean the ratio  $\|\vec{d}\|/\|\vec{\ell}\|$  of embedded data vs. dedicated space, but unlike  $\text{PoCS}_\phi$ , it allows for efficient updates of the embedded data. The third scheme we introduce is called PoR and is intended to be used as a PoR. Our new proof systems are derived from the [16] PoS  $\text{PoS}_o$  by replacing its underlying mode  $E_{\text{PoS}_o}$  by another mode of operation  $E_{\text{PoXX}} \in \{E_{\text{PoCS}_\phi}, E_{\text{PoCS}_F}, E_{\text{PoR}}\}$ . These modes take as input  $\chi$  (just like  $E_{\text{PoS}_o}$ ), and additionally some data  $\vec{d} = \{d_i\}_{i \in V_C}$  and output a file  $\vec{\ell} := E_{\text{PoXX}}(\chi, \vec{d})$  to be stored. The data  $\vec{d}$  can be recovered from  $\vec{\ell}$  at any time. These four modes are all illustrated in Figure 2.

## 1.4 Fisch’s PoR

That depth-robust graphs are useful to construct proofs of replication has been observed independently by Ben Fisch, he discusses this in a BPASE’18 talk<sup>3</sup> which took place almost a month before this paper was posted in a public archive.

In a recent paper [18] Fisch starts developing the foundations of PoR. His paper addresses many conceptual and technical aspects of PoR, including PoR constructions based on depth-robust graphs similar to the ones in this paper. His security proofs crucially use the main technical from this paper, i.e., that pebbling lower bounds for parallel time complexity imply lower bounds in the random oracle model as stated in Theorem 7. In a subsequent work Fisch presents PoR based on depth robust graphs with even better provable security guarantees [19]. The construction in that paper is conceptually somewhat different from ours in terms of the underlying graphs but also in the way in which the data is embedded. The graphs are stacked depth robust graphs, and are somewhat reminiscent of the simple and elegant PoS of Ren and Devadas [30] which is based on stacked expanders. We’ll say more on the efficiency of those constructions in §2.

## 1.5 Properties of PoS, PoCS and PoR

Fisch [18] observes that for any meaningful definition of PoR, a *PoR necessarily is also a PoS* as defined in [16]. Also a PoCS must necessarily be a PoS.

We (non-trivially) extend our security proof for  $\text{PoS}_o$  to prove that also the schemes  $\text{PoCS}_\phi$ ,  $\text{PoCS}_F$ , PoR are PoS (The final bound for  $\text{PoS}_o$ ,  $\text{PoCS}_\phi$ , PoR is stated in Corollary 8 in §8.2, the bound for  $\text{PoCS}_F$  is in the full version [29]. On the other hand, we observe that *being a PoS with the option to embed useful data is not sufficient to constitute a good PoCS or PoR*. Moreover the “whish list” of properties one might have for PoCS and PoR is somewhat contradictory. Our PoR candidate PoR is not a good PoCS, while  $\text{PoCS}_\phi$ ,  $\text{PoCS}_F$  do not make for a good PoR, as we’ll elaborate next.

The most important property we want from a PoCS is that any particular block of data from  $\vec{d} = \{d_i\}_{i \in V_C}$  cannot be recovered too efficiently from  $\vec{\ell}$ . The reason is that otherwise the PoCS wouldn’t compose: a malicious prover  $\tilde{P}$  could run a PoCS for some statement  $\chi$ ,

<sup>3</sup> [https://www.youtube.com/watch?v=8\\_90NpyRZEI](https://www.youtube.com/watch?v=8_90NpyRZEI)

and at the same time using the embedded data  $\vec{d}$  for a PoCS for other statement  $\chi'$ , thus pretending to dedicate more space than it does. To prevent this, we want the PoCS to *lock* the catalytic data, by which we mean accessing any particular block  $d_i \in \vec{d}$  should be almost as expensive as recovering the entire  $\vec{d}$  from  $\vec{\ell}$ .

In a typical application of a PoR, the data  $\vec{d}$  is not chosen by P but provided by V, together with some replication parameter  $r \in \mathbb{N}$  (and statement  $\chi$ ). P will then run the PoR for various statements  $\chi_1, \dots, \chi_r$  (generated from  $\chi$ ), each embedding  $\vec{d}$ . Informally, the security property we want is that a prover who later successfully passes the audits must have stored  $r$  *redundant* copies of  $\vec{d}$  (as  $\vec{d}$  can be incompressible, the redundancy requirement implies that PoR is a PoS). So unlike for a PoCS, in a PoR, being able to recover any data block efficiently is actually a feature, not an issue that breaks security.

We'll discuss those properties and how our schemes do or don't achieve them in more detail in §4, after having defined our various modes. We'll keep the discussion about the exact notion of a PoCS or PoR, and in particular any properties beyond being a PoS, like the locking and replication property mentioned above, informal. We expect future work to come up with the right definition for a PoCS, and also [18] is probably also not the last word on definitional issues for PoRs. There certainly are more properties one might need from a PoR or PoCS in particular applications that have not yet been identified, coming up with the right definitions and constructions satisfying them (in particular, showing that the constructions presented in this work do or do not satisfy them) is a promising research agenda.

## 2 Comparison With Previous Work

We somewhat divert from [16] when formally defining the security as a PoS. In [16], a PoS is defined to be  $(N_0, N_1, T)$ -secure if an adversary who stores a file  $\vec{\ell}$  of size  $N_0$  (recall that we measure size in blocks, typically of size something like  $w = 256$  bits) after the initialization phase, uses  $N_1$  space and  $T$  time during the proof executing phase, will fail in making the verifier accept with overwhelming probability. Below we shortly compare the four pebbling-based instantiations of the PoS<sub>o</sub> construction that so far have been suggested, and what security has been proven for them. Those just differ in the graphs  $\mathcal{G} = (V, E)$  and the dedicated set of challenge vertices  $V_C \subseteq V, |V_C| = N$ . We also mention the total number of edges  $|E|$ , as this basically determines the efficiency of the initialization procedure, and the indegree  $\delta$ , as this determines the size of the proofs and also the time to generate and verify proof. Let us mention that there is one work [3] constructing PoS using a completely different approach than graph-pebbling, for space reasons we'll only discuss his in the full version [29].

### 2.1 The original PoS [16]

[16] introduced the notion of PoS and gave two constructions, the first is

$$(\Theta(N/\log(N)), N/\log(N), \infty)\text{-secure with } |V| = N, \delta = 2, |E| = 2N$$

and based on a graph with high space pebbling complexity by Paul, Tarjan and Celoni [28], the second uses a rather sophisticated construction combining random bipartite graphs, superconcentrators and depth-robust graphs [17] and is

$$(\Theta(N), \infty, \Theta(N))\text{-secure with } |V| = N, \delta \in O(\log \log N), |E| \in O(N \log \log N)$$

## 2.2 The Ren-Devadas PoS [30]

Ren and Devadas [30] propose a very elegant instantiation of PoS<sub>o</sub> using stacked expanders and give a proof for it which in terms of security improves upon both constructions from [16] (just  $|E|$  is asymptotically larger). For any  $\alpha \in [0, 0.5]$ , their proof implies security (almost)

$$(\alpha \cdot N, (1 - \alpha) \cdot N, \infty)\text{-secure with } |V| \in O(N \log N), \delta \in O(1), |E| \in O(N \log N)$$

For say  $\alpha = 1/3$ , this means an adversary storing  $N_0 = N/3$  blocks after initialization, must use at least  $N_1 = 2N/3$  space during execution. Their construction is a stack of  $\log(N)$  expanders of indegree 2, and  $V_C$  is the graph on top of this stack.

## 2.3 Fisch’s tight PoS [19]

The PoS underlying Fisch’s recent PoR construction which we shortly discussed in §1.4, is based on a PoS that *for any*  $\epsilon > 0$  achieves

$$(N \cdot (1 - \epsilon), \infty, N)\text{-security with } |V| = O(N \log(1/\epsilon)), \delta \in O(1), |E| \in O(N \log(1/\epsilon))$$

Fisch provides concrete bounds for all the constants, the bounds are so good that he gets a very practical construction for parameters where the proofs guarantee good practical security. For our construction, which we discuss below, this is not the case. As there’s a huge gap between the lower and upper bounds on the security we can prove for PoS based on the simple depth robust graphs from [4, 6], it’s not clear whether the actual security of the “stacked” constructions as used in [19, 30] really is practically better, or if those constructions just allow for much tighter proofs, while not actually having better security in practice. Settling this is an interesting open question.

## 2.4 Our PoS

In this work we use the depth-robust graphs from [4, 6] to instantiate PoS<sub>o</sub>, and also our three new constructions which allow to embed useful data. *For any*  $\epsilon > 0$ , we can instantiate it as to get

$$(N \cdot (1 - \epsilon), \infty, N)\text{-security with } |V| = 4N, \delta \in O(\log(N)), |E| \in O(N \log N)$$

This might not seem terribly impressive, note that unlike [30] we don’t claim any lower bound on  $N_1$ , the space a cheating adversary must dedicate during proof execution. And asymptotically,  $|E|$  is larger than in the second construction of [16] which (ignoring constants) has the same security. But as we’ll explain next, we improve upon all existing constructions, except the subsequent work by Fisch [19], in three crucial points.

- 1. Unconditional Proof:** Our proof holds unconditionally (in the random oracle model), whereas [16, 30] only argued security against restricted adversaries who store a subset of the file an honest prover would store. Let us mention that for such relaxed adversaries, we can also prove bounds on the space a successful prover needs during execution.<sup>4</sup>

---

<sup>4</sup> Basically, for this restricted class of adversaries, whatever bound on time and/or space is proven for the underlying graph translates to a time and/or space bound for the construction. Our unconditional proof only translates parallel time complexity. The graphs we use to instantiate our construction are depth-robust, and such graphs are known [5] to have high “cumulative pebbling complexity”, which (for restricted adversaries as just mentioned) translates to the fact that if adversary runs in  $T$  rounds during proof execution, it must use  $\Omega(N^2/T)$  space on average during this computation.

2. **Tight Bound:** We get tight security: for any constant  $\epsilon > 0$ , we can instantiate our PoS to be  $((1 - \epsilon) \cdot N, \infty, N)$  secure. Equivalently, an adversary storing just an  $\epsilon$  fraction less than the honest prover, and which can run in time  $T = N$ , will still fail to make the verifier accept with overwhelming probability.

Having such a tight bound is crucial for many applications, as it means we get security against an adversary dedicating an  $(1 - \epsilon)$  fraction of the space for any  $\epsilon > 0$ . Note that even the (proof of the) [30] construction doesn't imply any security against adversary who dedicates just  $N_0 = N/2$ , i.e., half the claimed space.

3. **Security Against Parallelism:** The security we prove even holds if we strengthen the meaning of the parameter  $T$  from “total number of oracle queries”, to “total number of *parallel* oracle queries”, where each parallel query can contains many inputs, as long as in total they are bound by an exponential.

This stronger security notion implies that even massive parallelism doesn't help a potential adversary. This is useful in a setting where the timepoint at which audits happen is not known to the prover (in proofs of replication this can be achieved), and we have a bound on the latency of network between prover and verifier (so the prover cannot make  $T = N$  *sequential* computations in time less than this latency). Here we can be sure a prover who passes the audits really dedicates the claimed space, and does not simply reinitialize the entire space once the audit starts fast enough using massive parallelism. Compare this to the construction from [30], which can be initialized in sequential time  $\log(N)$  using parallelism  $N$ .

We will not use the formalism from [16] to quantify security outside of this subsection, but in our security statements explicitly state what is achieved, which should be easier to parse. The PoS security of  $E_{\text{PoS}_0}$  is stated in Corollary 8, The PoS security of  $E_{\text{PoCS}_\phi}$  and  $E_{\text{PoR}}$  in Corollary 13 and the PoS security of  $E_{\text{PoCS}_F}$  in the full version [29].

### 3 Basic Notation and Definitions

#### 3.1 Notation

For an object  $X$ ,  $\|X\|$  denotes its bitlength, for a set  $\vec{x}$ ,  $|\vec{x}|$  is the number of elements in  $\vec{x}$ . For an integer  $m$  we denote  $[m] \stackrel{\text{def}}{=} \{1, 2, \dots, m\}$  and for  $a, b \in \mathbb{R}$  we denote  $[a, b] \stackrel{\text{def}}{=} \{c : a \leq c \leq b\}$ . With  $\{0, 1\}^{\leq m}$  we denote the set of strings of length  $\leq m$ .

We typically use small greek letters  $\iota, \delta, \omega, \mu, \nu, \epsilon, \dots$  for our parameters used to quantify security, efficiency etc.. An exception is  $N$  which throughout denotes the space requirement of a prover. All these parameters are values in  $\mathbb{N}$  except  $\epsilon$  which is in  $[0, 1]$ . For the security games considered in this paper we use capital greek letters  $\Phi, \Lambda$ . The sans-serif font is used for interactive systems like parties  $V, P, \tilde{P}, A$  (modelled as randomized interactive Turing machines), functions  $H, F, g, f$  or algorithms like commitments discussed below ( $V, P$  and  $\tilde{P}$  are reserved for an honest verifier, an honest prover, and a potentially malicious prover, respectively). We use bold letters  $\vec{\ell}, \vec{d}, \vec{o}, \vec{c}, \dots$  for sets (usually ordered) of values, except for graph notation where we use simply  $\mathcal{G} = (V, E)$  to denote a graph with vertices  $V$  and directed edges  $E$ .

We will often consider a subset  $V_C \subset \mathbb{N}, |V_C| = N$  of challenge nodes. It will be convenient to define concise notion for mapping  $V_C$  to  $[N]$ , which we do using a tilde, i.e.,

$$V_C = (v_1, \dots, v_N) \quad \Rightarrow \quad (\tilde{v}_1, \dots, \tilde{v}_N) = (1, \dots, N) \quad (1)$$



## 3.2 Random Oracles

### 3.2.1 Fresh Random Oracles.

If  $H$  is a fixed random oracle and  $z \in \{0, 1\}^*$ , we denote with  $H_z$  the function  $H_z(\cdot) = H(z, \cdot)$ . If  $z$  is random and long enough (concretely, the amount of non-uniform advice an adversary has on  $H$  is not too large exponential in  $\|z\|$ ), we can treat  $H_z$  as a fresh uniformly random oracle [14]. We do this repeatedly in this work without always explicitly mentioning it.

### 3.2.2 The Parallel Random Oracle Model.

We prove security of our schemes in the *parallel random oracle model*, where in every round an adversary can output a set  $x_1, \dots, x_i$  of queries, and it receives the outputs  $H(x_1), \dots, H(x_i)$  at the beginning of the next round. For us the number of rounds will be important, but the total number of queries is secondary. Although also the total number of queries must be bound, in our proofs it can be as large as exponential in the block size  $w$ , and for the basic PoS<sub>o</sub>, the number of queries during the initialization phase can be even unbounded (not so for the other schemes). We will denote the number of oracle queries to  $H$  an adversary is allowed to make in the initialization and proof execution phase by  $q_1^H$  and  $q_2^H$ , respectively.

## 3.3 Commitments

We will make extensive use of a Merkle-tree commitment scheme, which allows to compute a short commitment to a long string, and later efficiently open any particular location of that long string. It is specified by a triple of algorithms `commit`, `open`, `verify` which use a hash-function  $H : \{0, 1\}^* \rightarrow \{0, 1\}^w$  as a building block. For the security as a commitment scheme, it's sufficient for  $H$  to be collision resistant. In our proofs we will sometimes need to extract committed values from the committing party, for this we must assume  $H$  is given as an oracle so our reduction can observe all queries.

If it's relevant what hash function is used it's shown as superscript (otherwise one can just assume any collision-resistant hash function is used). A party  $A$  which wants to commit to values  $\vec{x} = (x_1, \dots, x_m)$  invokes

$$(\phi_{\vec{x}}, \phi_{\vec{x}}^+) := \text{commit}^H(\vec{x})$$

here  $\phi_{\vec{x}}^+ \in \{0, 1\}^{(m-1)w}$  denotes the values of all inner nodes in the Merkle-tree, which are required to later efficiently open any position in  $\vec{x}$ , and  $\phi_{\vec{x}} \in \{0, 1\}^w$  is the value at the root, which is the commitment.

Once  $A$  announces  $\phi_{\vec{x}}$  it is committed to  $\vec{x}$ . It can then open any subset  $\vec{i} \subseteq [m]$  of the committed values (i.e.,  $\{x_i\}_{i \in \vec{i}}$ ) by invoking

$$\vec{o} := \text{open}^H(\vec{x}, \phi_{\vec{x}}^+, \vec{i}) \in \{0, 1\}^{\leq |\vec{i}| \lceil \log(m) \rceil \cdot w}.$$

Everyone can verify that  $\vec{o}$  is the correct opening by invoking  $\text{verify}^H(\phi_{\vec{x}}, \vec{i}, \vec{o})$  and accepting iff this value is 1.

## 3.4 Random Strings are Incompressible

In our proofs we'll repeatedly use the following fact, which states that a random string cannot be compressed.



► **Fact 1** (statement from [13]). *For any randomized encoding procedure  $\text{enc} : \{0, 1\}^r \times \{0, 1\}^n \rightarrow \{0, 1\}^m$  and decoding procedure  $\text{dec} : \{0, 1\}^r \times \{0, 1\}^m \rightarrow \{0, 1\}^n$  where*

$$\Pr_{x \leftarrow \{0, 1\}^n, \rho \leftarrow \{0, 1\}^r} [\text{dec}(\rho, \text{enc}(\rho, x)) = x] \geq \delta$$

we have  $m \geq n - \log(1/\delta)$ .

## 4 Overview of Our Modes and Protocols

In this section we will formally define the mode  $\mathbf{E}_{\text{PoS}_o}$  underlying the PoS from [16] and our new modes  $\mathbf{E}_{\text{PoR}}, \mathbf{E}_{\text{PoCS}_\phi}, \mathbf{E}_{\text{PoCS}_F}$  as illustrated in Figure 2. The actual protocols using those modes as illustrated in Figure 1 will then be defined in §4.5. As we define the modes, we will also continue our discussion from §1 showing how they (fail to) perform as PoCS or PoR. In particular, we'll show that our mode  $\mathbf{E}_{\text{PoR}}$  is not locking (and thus not suitable as PoCS), whereas  $\mathbf{E}_{\text{PoCS}_\phi}, \mathbf{E}_{\text{PoCS}_F}$  do not imply replication. All the modes are defined over a DAG  $\mathcal{G} = (V, E)$ , for  $i \in V$  we denote with  $\text{parents}(i) = \{j : (j, i) \in E\}$  the parents of  $i$ , and we define  $\vec{\ell}_{\text{parents}(i)} = \{\ell_j : j \in \text{parents}(i)\}$ .

### 4.1 The Mode $\mathbf{E}_{\text{PoS}_o}$

In the basic PoS the file  $\vec{\ell} := \mathbf{E}_{\text{PoS}_o}(\chi)$  contains the “labels” of nodes in  $V_C$ , where the labels of the nodes of the underlying DAG  $\mathcal{G} = (V, E)$  are computed in topological order by hashing (using a fresh random oracle sampled using  $\chi$ ) the labels of its parents

$$\forall i \in V : \ell_i = \mathbf{H}_\chi(i, \vec{\ell}_{\text{parents}(i)}) \quad , \quad \vec{\ell} \stackrel{\text{def}}{=} \{\ell_i\}_{i \in V_C} \quad (\mathbf{E}_{\text{PoS}_o}) \quad (2)$$

The most obvious way to somehow embed data  $\vec{d} = \{d_i\}_{i \in [N]}$  into this basic PoS is to simply XOR the data blocks to the labels in  $V_C$ . There are two natural ways to do this, XOR the data to the labels as the computation goes on, or first compute the labels and then XOR the data to it. The first approach is basically what we do in our construction PoR, and the second in  $\text{PoCS}_\phi$ . Before computing the labels, we first commit to  $\vec{d}$ , and then sample a fresh random oracle to compute the labels using this commitment. We'll explain below why without this trick our constructions would miserably fail to be PoS.

### 4.2 The Mode $\mathbf{E}_{\text{PoR}}$

In our PoR  $\vec{\ell} := \mathbf{E}_{\text{PoR}}(\chi, \vec{d})$ , the data  $\vec{d} = \{d_i\}_{i \in [N]}$  is first committed

$$(\phi_{\vec{d}}, \phi_{\vec{d}}^+) := \text{commit}^{\text{H}_\chi}(\vec{d}) .$$

Then  $\phi_{\vec{d}}$  is used to sample a fresh random oracle  $\mathbf{H}_{\chi, \phi_{\vec{d}}}(\cdot) = \mathbf{H}_\chi(\phi_{\vec{d}}, \cdot)$ , which is then used to compute the labels. For labels of a node  $i \in V_C$ , one additionally XORs the data block  $d_i$  (recall that  $\{i\}_{i \in V_C} = [N]$ ) to the label right after it has been computed.

$$\ell_i = \begin{cases} \mathbf{H}_{\chi, \phi_{\vec{d}}}(i, \vec{\ell}_{\text{parents}(i)}) & \text{if } i \in V \setminus V_C \\ \mathbf{H}_{\chi, \phi_{\vec{d}}}(i, \vec{\ell}_{\text{parents}(i)}) \oplus d_i & \text{if } i \in V_C \end{cases} \quad , \quad \vec{\ell} = \{\ell_i\}_{i \in V_C} \quad (\mathbf{E}_{\text{PoR}}) \quad (3)$$

### 4.3 The Mode $\mathbf{E}_{\text{PoCS}_\phi}$

In our PoCS  $\vec{\ell} := \mathbf{E}_{\text{PoCS}_\phi}(\chi, \vec{d})$  one first computes  $\phi_{\vec{d}}$  as above, uses this to sample a fresh random oracle  $\mathbf{H}_{\chi, \phi_{\vec{d}}}$  to compute labels (as in the basic PoS<sub>o</sub>), and only then XORs the data to the labels to be stored

$$\forall i \in V : \ell_i = \mathbf{H}_{\chi, \phi_{\vec{d}}}(i, \vec{\ell}_{\text{parents}(i)}) \quad (4)$$

$$\forall i \in V_C : \ell'_i = \ell_i \oplus d_i, \quad \vec{\ell} = \{\ell'_i\}_{i \in V_C} \quad (\mathbf{E}_{\text{PoCS}_\phi}) \quad (5)$$

As mentioned above, the fact that the random oracle  $\mathbf{H}_{\chi, \phi_{\vec{d}}}$  used to compute the labels depends on the commitment  $\phi_{\vec{d}}$  to  $\vec{d}$  is crucial, let us sketch why. Assume we'd change  $\mathbf{H}_{\chi, \phi_{\vec{d}}}$  to  $\mathbf{H}_\chi$  in the definition of  $\mathbf{E}_{\text{PoCS}_\phi}$ . Now a malicious prover (in the protocol  $\text{PoCS}_\phi$  to be defined in §4.5) could set the  $\ell'_i$  to be stored to whatever it wants (and thus also store them using low space). Only after choosing the  $\ell'_i$ , it then fixes the data  $\vec{d} = \{d_i\}_{i \in [N]}$  by “equivocating” it, i.e., setting it as  $d_i := \ell'_i \oplus \ell_i$ , so everything is consistent. This malicious behaviour (not using any space) cannot be distinguished from honest behaviour, thus it's not a PoS.

Now let us observe that  $\text{PoCS}_\phi$  is not a good PoR, as it doesn't imply replication. A prover who is supposed to compute and store  $\vec{\ell}_i := \text{PoCS}_\phi(\chi_i, \vec{d})$  for  $r$  statements  $\chi_1, \dots, \chi_r$  but the same  $\vec{d}$  can instead store  $\vec{d}$  once in the clear, and for then for each  $\chi_j$  only store the labels  $\{\ell_i\}_{i \in V_C}$  as in eq.(4) instead  $\{\ell'_i = \ell_i \oplus d_i\}_{i \in V_C}$ , i.e., avoid the XORing step of eq.(5). Note that this prover has only stored one copy of  $\vec{d}$ , instead of storing it  $r$  times redundantly, while it can still pass the audits for all  $\chi_i, i \in [r]$  because it can compute the correct labels  $\ell'_i$  using its single copy of  $\vec{d}$  as  $\ell'_i = \ell_i \oplus d_i$ . The prover here doesn't seem to gain much, in particular it doesn't save on space by deviating from the honest behaviour. But in a PoR we probably want to enforce replication, or at least argue that it's not rational for a prover to deviate, and there are settings where deviating as just explained can be rational. Assume the prover has large remote storage space, but only low bandwidth to access it. By deviating as explained, it can use the space for the  $r$  proofs without large communication, in particular, without ever having to send  $\vec{d}$  to the remote disk.

In the other direction one can argue that  $\mathbf{E}_{\text{PoR}}$  is not a good PoCS as given all labels  $\{\ell_i\}_{i \in V}$  as in eq.(3), one can efficiently recover the embedded data as  $d_i = \ell_i \oplus \mathbf{H}_{\chi, \phi_{\vec{d}}}(i, \vec{\ell}_{\text{parents}(i)})$ , so it doesn't provide the locking property we want from a PoCS. The above argument highlights a problem with the PoCS security of  $\mathbf{E}_{\text{PoR}}$ , but is not totally convincing, as the prover actually only needs to store the  $\ell_i$  for  $i \in V_C$  (not all  $i \in V$ ), so it couldn't necessarily recover those labels efficiently.

### 4.4 The Mode $\mathbf{E}_{\text{PoCS}_F}$

Besides  $\text{PoCS}_\phi$ , we propose a second PoCS  $\text{PoCS}_F$ , which allows for efficient updates. Instead of committing to  $\vec{d}$  and using this commitment to sample the random oracle  $\mathbf{H}_{\chi, \phi_{\vec{d}}}$  for computing the labels as in  $\text{PoCS}_\phi$ , in  $\text{PoCS}_F$  the labels are computed directly using  $\mathbf{H}_\chi$ , i.e., independently of  $\vec{d}$ . To prevent the “equivocation” attack outlined above, in  $\text{PoCS}_F$  the prover samples (using  $\chi$ ) a random invertible function  $\mathbf{F}_\chi : \{0, 1\}^{(\kappa+1) \cdot w} \rightarrow \{0, 1\}^{\kappa \cdot w}$  and applies it to the data before XORing it to the label, where  $\kappa$  is a parameter discussed below. The labels  $\ell_i, i \in V \setminus V_C$  have length  $w$  bits, the labels  $\ell_i, i \in V_C$  are  $(\kappa + 1) \cdot w$  bits long. Below  $\mathbf{H}_\chi : \{0, 1\}^{\leq \iota} \rightarrow \{0, 1\}^{(\kappa+1) \cdot w}$ , and  $\mathbf{H}_\chi(\cdot)|_w$  means we cap the output after  $w$  bits.

$$\ell_i = \begin{cases} \mathbf{H}_\chi(i, \vec{\ell}_{\text{parents}(i)}) & \text{if } i \in V_C \\ \mathbf{H}_\chi(i, \vec{\ell}_{\text{parents}(i)})|_w & \text{if } i \in V \setminus V_C \end{cases} \quad (6)$$

$$\forall i \in V_C : \ell'_i = \ell_i \oplus \mathbf{F}_\chi(d_i), \quad \vec{\ell} = \{\ell'_i\}_{i \in V_C} \quad (\mathbf{E}_{\text{PoCS}_\phi}) \quad (7)$$

A random invertible function  $F_\chi$  as used in this construction can be constructed efficiently, and with almost no loss in concrete security (which is crucial for our application) instantiated from a random oracle [21].

PoCS $_\phi$  has a better rate than PoCS $_F$ . In PoCS $_\phi$  we have rate  $\frac{\|\vec{d}\|}{\|\vec{\ell}\|} = 1$ , so the stored file  $\vec{\ell}$  is as big as the data  $\vec{d}$  that can be recovered from it.<sup>5</sup> The rate of PoCS $_F$  is only  $\frac{\|\vec{d}\|}{\|\vec{\ell}\|} = \frac{\kappa}{\kappa+1}$ . For efficiency reasons the  $\kappa$  can't be too large ( $\kappa \approx 10$  is reasonable, then the size of  $\vec{\ell}$  is  $\approx 10\%$  larger than  $\vec{d}$ ).

But unlike PoCS $_\phi$ , PoCS $_F$  allows for fast updates of the catalytic data: if P wants to change a single data block  $d_i$  (embedded in  $\ell'_i = \ell_i \oplus F_\chi(d_i)$ ) to  $d'_i$  then it can simply replace the label  $\ell'_i$  with  $\ell'_i \oplus F_\chi(d_i) \oplus F_\chi(d'_i)$ . It then must also update the commitment  $(\phi_\ell, \phi_\ell^\perp)$ , but this takes only time  $\log(N)$ . For this update P actually needs to know the currently embedded data block  $d_i$ . There are natural settings where  $\vec{d}$  is readily available in the clear. For example if the catalytic data  $\vec{d}$  encoded in  $\vec{\ell}$  is used as backup, and a working copy of  $\vec{d}$  is available in the clear. So we think this feature might be useful. This mode is somewhat different than the other three modes considered. Below we define and analyze the PoS $_o$ , PoR, PoCS $_\phi$  modes together as they're very similar, but PoCS $_F$  is different, and for space reasons the precise definitions and security proofs are only given in the full version [29].

► Remark (efficiently updatable PoR). Looking at Figure 2, one might wonder why there's no mode E $_{\text{PoR}_F}$ , where the data  $\vec{d}$  is first pre-processed by  $F_\chi$  as in E $_{\text{PoCS}_F}$ , but then XORed to the labels right after they are computed (as in E $_{\text{PoR}}$ ). The reason is that the only advantage of preprocessing  $\vec{d}$  using  $F_\chi$  as in E $_{\text{PoCS}_F}$  instead of committing to it as in E $_{\text{PoCS}_\phi}$  is the fact that it makes updating data blocks cheap. But if we XOR the data to the labels right after it has been computed as in E $_{\text{PoR}}$ , then updating a block in label  $\ell_i$ , will also change all subsequent labels  $\ell_j, j > i$ , even if the hash function used to compute labels is independent of  $\vec{d}$ . Thus, this update is not cheap after all. It's an interesting open problem to construct a candidate for a PoR where data blocks can be efficiently updated.

## 4.5 The Protocols PoS $_o$ , PoCS $_\phi$ and PoR

The protocols we consider in this work are a generalization of the pebbling-based PoS from [16] PoS $_o$ , where we allow the prover to chose and embed additional data  $\vec{d}$  into the file  $\vec{\ell}$  to be stored. We define PoS $_o$ , PoCS $_\phi$ , PoR together as they are very similar, we use PoXX  $\in \{\text{PoS}_o, \text{PoCS}_\phi, \text{PoR}\}$  as placeholder for any of those constructions.

$w, \mu$  : A block length  $w$  ( $w = 256$  is a typical value) and a statistical security parameter  $\mu$ .

$\mathcal{G}$  : A directed acyclic graph  $\mathcal{G} = (V, E)$  with max. indegree  $\delta$  and a designated set  $V_C \subseteq V$  of "challenge nodes" of size  $N = |V_C|$ .

$H$  : A hash function, which for the proof is modelled as a random oracle  $H : \{0, 1\}^{\leq \iota} \rightarrow \{0, 1\}^w$  which takes inputs of length at most  $\iota = (\delta + 2) \cdot w$  bits.

The space required by the honest prover is  $\approx N \cdot w = |V_C| \cdot w$  bits (we'll discuss the exact space requirement in Remark 4.7).

<sup>5</sup> The prover also must store opening information  $\phi_\ell^\perp$  for a Merkle commitment, but as we'll discuss in Remark 4.7 this is small compared to  $\vec{\ell}$ .

## 4.6 Initialization

V picks a random statement  $\chi \in \{0, 1\}^w$  and sends it to P.

If  $\text{PoXX} \neq \text{PoS}_o$ , the prover P can choose any data  $\vec{d} = \{d_i\}_{i \in [N]}$ ,  $d_i \in \{0, 1\}$  and then computes the file to store (if  $\text{PoXX} = \text{PoS}_o$ ,  $\vec{d}$  is empty)

$$\vec{\ell} := E_{\text{PoXX}}(\chi, \vec{d})$$

as shown in Figure 2 and explained in eq.(2)-(4).

Finally P computes the commitment  $(\phi_{\vec{\ell}}, \phi_{\vec{\ell}}^{\pm}) := \text{commit}(\vec{\ell})$  for all the labels  $i \in V_C$ , sends the short commitment  $\phi_{\vec{\ell}}$  to V and locally stores the opening information  $\phi_{\vec{\ell}}^{\pm}$ . This concludes the initialization if we assume P is honest during this phase (we'll discuss the general case in Remark 4.7).

At the end of the initialization phase the verifier stores the short strings  $\chi, \phi_{\vec{\ell}}$ . The prover stores  $\chi, \phi_{\vec{\ell}}^{\pm}$  and additionally a large file  $\vec{\ell} = \{\ell_i\}_{i \in V_C}$  of size  $\|\vec{\ell}\| = w \cdot N$ .

## 4.7 Proof execution

The protocol where  $P(\vec{\ell}, \chi, \phi_{\vec{\ell}}^{\pm})$  convinces  $V(\chi, \phi_{\vec{\ell}})$  that it stores  $\vec{\ell}$  is very simple. V samples a few nodes from the challenge set  $\vec{c} = (c_1, \dots, c_{\mu}) \subset V_C$  at random, and sends the challenge  $\vec{c}$  to P. P sends openings  $\vec{\sigma} := \text{open}(\vec{\ell}, \phi_{\vec{\ell}}^{\pm}, \vec{c})$  to the labels  $\{\ell_i\}_{i \in \vec{c}}$  to V, who then accepts iff  $\text{verify}(\phi_{\vec{\ell}}, \vec{c}, \vec{\sigma}) = 1$ .

► **Remark (prover is honest during initialization).** For most of the paper we will assume that even a malicious prover  $\tilde{P}$  follows the protocol during the initialization phase (i.e., behaves like the honest P). Of course we can't make this assumption in practice, that's why pebbling-based PoS have an extra communication round at the end of the initialization phase where – for some statistical security parameter  $\nu$  – V challenges  $\tilde{P}$  to open  $\nu$  labels and their parents to check that they were correctly computed. For this,  $\tilde{P}$  initially sends a commitment to all nodes  $V$ , not just  $V_C$ , and (except for  $\text{PoS}_o$ ) also a commitment to  $\vec{d}$ . If  $\tilde{P}$  committed to labels  $\{\ell_i^*\}_{i \in V}$  where it cheated on an  $\epsilon$  fraction of the labels, i.e., for  $\text{PoS}_o$  this means we have  $\ell_i^* \neq H_{\chi}(i, \ell_{p_1}^*, \dots, \ell_{p_{\delta_i}}^*)$ , then  $\tilde{P}$  will fail to pass this check with probability  $1 - (1 - \epsilon)^{\nu}$ .  $\tilde{P}$  can still get away with cheating at a small fraction of labels, but one can easily take care of this in the proof by assuming that storing such inconsistent labels can be done by P “for free”. As this is a minor technicality in the proof, we ignore this as not to obfuscate the main technical contributions.

► **Remark (P's space).** The size of the file  $\vec{\ell}$  is  $N \cdot w$  bits, which is basically the same as the  $(N - 1) \cdot w$  bits required to store the opening info  $\phi_{\vec{\ell}}^{\pm}$  of the Merkle-tree commitment to  $\vec{\ell}$ : on the 0'th level of the tree (the leaves) we have the  $N$  labels, then on level 1 we have  $N/2$  values, on level 2 we have  $N/4$  values, etc., for a total of  $N/2 + N/4 + \dots + 2 + 1 = N - 1$  labels of internal nodes that constitute  $\phi_{\vec{\ell}}^{\pm}$ . But the prover can decide to not store levels  $1 \dots k$ , thus saving only  $(N - 1)/2^k$  blocks, while all values in the omitted layers can be recomputed by hashing at most  $2^k$  leaf values (i.e., values from  $\vec{\ell}$ ). Thus, for say  $k = 5$ , the Merkle tree requires just a  $1/32$  fraction of the space of  $\vec{\ell}$ , but requires to hash 32 values. In practice that wouldn't be expensive as those leaf labels can always be stored consecutively on a disk, and thus reading them comes at small cost compared to reading the first random block (and hashing 32 blocks is not expensive compared to a disk access). In the discussions in this writeup we will thus mostly ignore the space required for storing the opening information  $\phi_{\vec{\ell}}^{\pm}$ .

## 5 The Main Proof Ideas

In this section we’ll discuss the main ideas used in the proofs of this paper, and give an overview of the work we borrowed ideas from. Pebbling is a game played on directed acyclic graphs (DAG), where a player can put pebbles on nodes of the graph according to some rules, and its goal is usually to pebble some particular node or set of nodes using as few “resources” as possible. There are various pebbling games one can consider, in this work we just consider the basic black-pebbling game, where the player can put a pebble on a node if all of its parents have pebbles. The resource considered is typically time (i.e., how many rounds it takes) or space (i.e., the maximum number of pebbles on the graph at any time), or combinations thereof. For example cumulative space (the sum of the number of pebbles on the graph over all rounds) [5] and sustained space (the number of rounds at which many pebbles were on the graph) [6] have been suggested to model memory-hard functions. Another important distinction is between sequential and parallel strategies; a parallel player can – in every round – put as many pebbles on the graph as he wants, whereas a sequential player can put only one. For reasons discussed below, in this paper we will consider **time complexity** in the **parallel black-pebbling** model.

As pebbling is a simple combinatorial game, it’s often possible to prove unconditional lower bounds on the resources required to pebble some graphs, and in some cases one can prove that these bounds imply lower bounds for problems in more interesting computational models. In particular, if the pebbling game considered is “deterministic” in the sense that the player is initially given the graph and a designated set of nodes to pebble, then one can use an elegant proof strategy (coined “ex-post facto” in the paper [15] that introduced it) to translate basically any pebbling lower bound to a corresponding lower bound in the random oracle model for computing the “labels” of the designated set of nodes, where the label of a node is the output of the random oracle on input the labels of its parents.<sup>6</sup>

Pebbling games capture most constructions of so called memory-hard functions (MHFs), which are functions that require a lot of memory to be computed. One distinguishes between *data-independent* MHFs, where the memory access pattern is independent of the functions input, and more general *data-dependent* MHFs. The pebbling game capturing data-independent MHFs is deterministic, but for data-dependent MHFs it’s randomized, for this reason almost all security proofs for data-dependent MHFs need to make additional assumptions on the adversary. An exception is the recent security proof for the data-dependent MHF called SCRYPT [7], which proves that SCRYPT has high cumulative memory complexity in the parallel random-oracle model. Despite the fact that here the underlying pebbling game is randomized, their proof does not need to make any assumptions on the adversarial behaviour.

Like data-dependent MHFs, also the pebbling game (called  $\Phi$  and defined in §6.1) underlying pebbling-based PoS [16, 30] is randomized. Prior to this work no pebbling-based PoS had an unconditional security proof in the random oracle model; one had to assume that an adversarial prover only stores a subset of the data the honest prover would store, but not arbitrary functions of this data.<sup>7</sup>

<sup>6</sup> The high-level idea of an “ex-post-facto” proof is to look at the execution of the game in the random oracle model and translate this to a pebbling strategy, where every time a label is computed, we put a pebble on the corresponding node. Now, if the resources (where a round the pebble game translates to a round of queries to the random oracle, and a pebble translates to space required to store one label) required by the labelling game are smaller than in the derived pebbling game, we can use the adversary in the labelling game to compress the random oracle. But this is impossible as a uniformly random string cannot be compressed, so we have a contradiction, and the labelling game must have used at least as many resources as the lower bound for the corresponding pebbling game dictates.

<sup>7</sup> In [16] some combinatorial conjectures were stated which – if true – would have implied that restricting

The key observation that allows us to translate pebbling-lower bounds for a “randomized” pebbling game like  $\Phi$  to lower bounds for the “labelling game”  $\Lambda_{\text{PoS}_o}$  in the random oracle model (this game is defined in §7.1) is already implicit in [7], and goes as follows: If the complexity we consider is *time* complexity in the *parallel* black-pebbling game, then the optimal pebbling strategy is oblivious to the randomness (which in our game is a random node we need to pebble). Concretely, the strategy minimizing the number of rounds required is to put in every round a pebble on every possible node (i.e., every node whose parents are pebbled). We observe that the reason “ex-post facto” proofs can’t be done for randomized pebbling games is that the adversaries’ pebbling strategy can depend on the game’s randomness, but as just outlined, for parallel time complexity we can assume the adversary is oblivious to the randomness, and this allows us to push through a pretty standard ex-post facto type proof (proof of Theorem 7 in §7.2) showing that lower bounds on the hardness of the game  $\Phi$  imply lower bounds on the game  $\Lambda_{\text{PoS}_o}$ , which captures the security of  $\text{PoS}_o$  as a PoS. Very informally, the proof is a compression argument, which uses an adversary that is “too successful” in computing the labels of nodes it is being challenged on into a compressing encoding algorithm for the random oracle  $H$ . As a random oracle is incompressible, such an encoding cannot exist, and we get a contradiction.

In Theorem 12 in §8.2 we extend this proof from the basic  $\text{PoS}_o$  to the modes  $\text{PoCS}_\phi$  and  $\text{PoR}$ . The problem we face is that now, the values this “too successful” adversary predicts are not just outputs (i.e., labels  $\ell_i$  as in  $\text{E}_{\text{PoS}_o}$ ) of the random oracle  $H$ , but now they are of the form  $\ell_i \oplus d_i$ , where  $d_i$  is chosen by the adversary itself. Thus we can’t readily use the fact that we learned them to compress  $H$ . To solve this problem, we use the fact that in  $\text{PoCS}_\phi, \text{PoR}$ , the adversary must first commit to the  $d_i$ ’s, and this commitment is then used to sample a fresh random oracle to compute the labels. We let our encoding algorithm first runs this adversary who chooses the  $d_i$ ’s and computes the commitment. From this adversary we can extract all the  $d_i$ ’s. Once these are known, the encoding proceeds basically as for the basic  $\text{PoS}_o$ .

Extending the proof to show that our efficiently updatable  $\text{PoCS}$   $\text{PoCS}_F$  is a PoS is much more challenging, and for space reasons we only provide the proof in the full version [29], now only giving a high level idea of the challenges. In  $\text{PoCS}_F$  the labels of the “too successful” adversary predicts are of the form  $\ell_i \oplus F(d_i)$ , but the adversary has not committed to the  $d_i$ ’s before computing the  $d_i$ ’s. The key idea is to replace in the security game the random invertible function  $F : \{0, 1\}^{\lambda-w} \rightarrow \{0, 1\}^\lambda$  with the composition of two randomly sampled functions  $g(f(\cdot))$ , where  $f : \{0, 1\}^{\lambda-w} \rightarrow \{0, 1\}^{w/2}$ ,  $g : \{0, 1\}^{w/2} \rightarrow \{0, 1\}^\lambda$ , and argue that with high probability this game will behave like the original one (in particular, the adversary is almost as successful here). In this new game, we can recover  $\ell_i$  from the labels the adversary predicts, which now are of the form  $\ell_i \oplus g(f(d_i))$ , if additionally given only the short  $w/2$  bit string  $f(d_i)$ , this is good enough to get compression and again push through an ex-post-facto type proof.

## 6 The Graph Pebbling Game $\Phi$ and its Hardness

In this section we define a pebbling game  $\Phi$  and show it’s hard if instantiated with depth-robust graphs. Later we will prove that hardness of  $\Phi$  implies hardness of games capturing the PoS security of our schemes.

---

adversaries like this is without loss of generality. But these conjectures have been beautifully refuted in [24].

## 6.1 The Pebbling Game $\Phi(\mathcal{G}, V_C)$

The game is parameterized by a DAG  $\mathcal{G} = (V, E)$ , a subset  $V_C \subseteq V$  of  $|V_C| = N$  challenge nodes, and an integer  $s, 0 \leq s \leq N$ . It is played by an adversary given as a pair  $\mathbf{A}_\Phi = \{\mathbf{A}_\Phi^1, \mathbf{A}_\Phi^2\}$ .

**initialization:**  $\mathbf{A}_\Phi^1$  gets no input, and outputs the initial pebbling configuration, which is a subset  $P_0 \subseteq V$  of  $|P_0| = s$  nodes.

**execution:** A random challenge node  $c \leftarrow V_C$  is chosen.

$\mathbf{A}_\Phi^2$  gets as input  $P_0$  and the challenge  $c$ . It then proceeds in rounds, starting at round 1.

In round  $i$ ,  $\mathbf{A}_\Phi^2$  can place (arbitrary many) pebbles on the nodes of  $V$  to update the pebbling configuration from  $P_{i-1}$  to  $P'_i$  according to the following rule: a pebble can be placed on node  $v \in V$  only if all the parents of  $v$  are pebbled in  $P_{i-1}$ . It then can remove any number of pebbles to get the configuration  $P_i \subseteq P'_i$ .

► **Definition 2** (hardness of the game  $\Phi$ ). For  $s, t \in \mathbb{N}, \epsilon \in [0, 1]$ , we say  $\mathbf{A}_\Phi$  does  $(s, t, \epsilon)$ -win the pebbling game  $\Phi(\mathcal{G}, V_C)$  (as defined above) if the probability (over the choice of  $c$  and  $\mathbf{A}_\Phi$ 's random coins) that  $\mathbf{A}_\Phi^2$  puts a pebble on  $c$  in  $t - 1$  rounds or less is at most  $\epsilon$ .

We say  $\Phi(\mathcal{G}, V_C)$  is  $(s, t, \epsilon)$ -hard if no such  $\mathbf{A}_\Phi$  exists, that is, no adversary can pebble an  $\epsilon$  fraction of  $V_C$  in  $t$  rounds or less, having only  $s$  initial pebbles.

► **Remark** (greedy is best). We observe that the optimal strategy for  $\mathbf{A}_\Phi^2$  is trivial: the greedy strategy, where in every round  $\mathbf{A}_\Phi^2$  puts pebbles on all nodes possible and never removes a pebble, is at least as good as any other strategy. This greedy strategy is oblivious to the challenge  $c$ , which will be crucial in our proofs.

## 6.2 Depth Robust Graphs

► **Definition 3** (depth-robust graphs). A DAG  $\mathcal{G} = (V, E)$  on  $V = |N|$  nodes is  $(e, d)$ -depth robust if after removing any subset of  $e \cdot N$  nodes, there remains a path of length  $d \cdot N$ .

Such graphs were first considered by Erdős et al. [17], and recent work has made them more practical, cf. [4, 6] and references therein. Concretely, for any  $\epsilon > 0$ , [6] constructs a family  $\{\mathcal{G}_N^\epsilon\}_{N \in \mathbb{N}}$  of graphs of indegree  $O(\log N)$  (here the hidden constant depends on  $\epsilon$ ) which, for any  $e, d, e + d \leq 1 - \epsilon$  are  $(e, d)$ -depth robust.

Note that any graph, even the complete graph (which has indegree  $N - 1$ ) is only  $(e, d)$  depth-robust for  $e + d = 1$ . It is maybe surprising that one gets almost as good depth-robustness as the complete graph with only  $O(\log N)$  indegree (on the negative side, it's known that  $\Omega(\log N)$  indegree is necessary for this). Let us mention that the indegree of the  $\mathcal{G}$  we use to instantiate our schemes is important as the efficiency of our schemes (in particular the proof size) depends linearly on it.

## 6.3 $\Phi(\mathcal{G}, V_C)$ is Hard if $\mathcal{G}$ is Depth Robust

Let us observe that the  $\Lambda_{\text{PoS}_0}(\mathcal{G}, V_C)$  cannot be  $(s = N \cdot c_e, t, \epsilon = c_e)$ -hard for any  $c_e \in [0, 1]$  even for tiny  $t = 1$ , as one always can simply put those  $s$  initial pebbles on an  $c_e$  fraction of  $V_C$ , and this  $\epsilon = c_e$  fraction is then already pebbled in round 1. By the lemma below, using the depth-robust graphs  $\mathcal{G}_{4N}^{\epsilon'}$  the game becomes hard – i.e. we need  $t \geq N$  rounds – for just a slightly larger fraction  $\epsilon = c_e + 4\epsilon'$ .

► **Lemma 4** (hardness of  $\Phi$  with the depth-robust graphs from [6]). For any  $N \in \mathbb{N}, \epsilon' > 0$  consider the graph  $\mathcal{G}_{4N}^{\epsilon'}$  from [6], which is  $(e, d)$ -depth robust for any  $e + d \geq 1 - \epsilon'$ . Let  $V_C \subset V, |V_C| = N$  be the  $N$  topologically last nodes in  $V$ . Then, for any  $c_e \in [0, 1]$  the game  $\Phi(\mathcal{G}_{4N}^{\epsilon'}, V_C)$  is  $(s, t, \epsilon)$ -hard with

$$s = N \cdot c_e \quad , \quad t = N \quad , \quad \epsilon = c_e + 4\epsilon'$$



**Proof.** We can assume  $\epsilon' \leq 1/4$  as the statement is void for  $\epsilon < 0$ . Let  $e = c_e/4$ , then  $s = e \cdot 4N$  and  $e \leq 1/4$  (as  $c_e \leq 1$ , which holds as for  $s \geq N$  the statement is void).  $e+d \geq 1-\epsilon'$  implies  $d \geq 1 - e - \epsilon'$ . After removing  $s$  nodes (i.e., an  $e$  fraction) from  $V$ , by the  $(e, d)$  depth-robustness, there's still a path  $P \subset V$  of length at least  $d \cdot 4N \geq 4N(1 - e - \epsilon') \geq 2N$  in  $V \setminus S$  (second inequality using  $e \leq 1/4$  and  $\epsilon' \leq 1/4$ ). All but  $4N(e + \epsilon')$  of  $V_C$  must lie on this path (as the path contains all but  $4N(e + \epsilon')$  of the vertices), i.e.,

$$\frac{|V_C \cap P|}{N} \geq \frac{N - 4N(e + \epsilon')}{N} = 1 - 4e - 4\epsilon' = 1 - c_e - 4\epsilon'$$

The nodes in  $V_C \cap P$  are all at the end of the path  $P$  (as  $V_C$  was chosen topologically last in  $V$ ), and as  $|P| \geq 2N, |V_C| = N$ , each node in  $V_C \cap P$  has depth at least  $N$  in  $P$ , thus the number of sequential pebbling queries required to put a pebble on any of those nodes is  $t > N$ . Equivalently, only an  $\epsilon = 1 - \frac{|V_C \cap P|}{N} \leq c_e + 4\epsilon'$  fraction of  $V_C$  can be pebbled in  $t$  rounds or less, as claimed.  $\blacktriangleleft$

## 7 PoS Security of PoS<sub>o</sub>

In this Section we state and prove our main technical result Theorem 1, which states that hardness of the pebbling game  $\Phi$  implies hardness of a game  $\Lambda_{\text{PoS}_o}$  capturing the PoS security of PoS<sub>o</sub>. We start with defining the the  $\Lambda_{\text{PoS}_o}$  game

### 7.1 The Labelling Game $\Lambda_{\text{PoS}_o}(\mathcal{G}, V_C, w)$

The game is parameterized by a DAG  $\mathcal{G} = (V, E)$ , a subset  $V_C \subseteq V$  of  $|V_C| = N$  challenge nodes and a block size  $w$ . Moreover a function  $H_* : \{0, 1\}^{\leq \iota} \rightarrow \{0, 1\}^w, \iota = (\delta + 2) \cdot w$ . Let

$$\forall i \in V : \ell_i = H_*(i, \vec{\ell}_{\text{parents}(i)}) \quad (8)$$

(note that these are the labels  $\text{PoS}_o(\chi)$  would compute if  $H_\chi = H_*$ ). The game is played by an adversary  $A_{\text{PoS}_o} = \{A_{\text{PoS}_o}^1, A_{\text{PoS}_o}^2\}$

**initialization:**  $A_{\text{PoS}_o}^1$  is given oracle access to  $H_*$ . It outputs a string (the initial state)  $S_0$  of length  $\|S_0\| = m$  bits ( $A_{\text{PoS}_o}^1$  is computationally unbounded).

**execution:** A random challenge node  $c \leftarrow V_C$  is chosen.

$A_{\text{PoS}_o}^2$  gets as input  $S_0$  and the challenge  $c$ . It then proceeds in rounds, starting at round 1.

In round  $i$ ,  $A_{\text{PoS}_o}^2$  gets as input its state  $S_{i-1}$ . It can either decide to stop the game by outputting a single guess  $\ell_{\text{guess}}$  (for  $\ell_c$ ), or it can make one parallel oracle query: on query  $(x_1, \dots, x_{q_i})$  it receives  $(y_1, \dots, y_{q_i})$  where  $y_i = H_*(x_i)$ . It can do any amount of computation before and after this query, and at the end of the round output its state  $S_i$  for the next round.

► **Definition 5** (hardness of the game  $\Lambda_{\text{PoS}_o}$ ). For  $m, t, q_2^H, w, \alpha \in \mathbb{N}$  and  $p_H, \epsilon, \epsilon' \in [0, 1]$ , we say  $A_{\text{PoS}_o} = (A_{\text{PoS}_o}^1, A_{\text{PoS}_o}^2)$  does  $(m, t, \epsilon, p, q_2^H)$ -win the labelling game  $\Lambda_{\text{PoS}_o}(\mathcal{G}, C, w)$  (as defined above) if for all but a  $p_H$  fraction of  $H_*$  the following holds: for at most an  $\epsilon$  fraction of challenges  $c$ ,  $A_{\text{PoS}_o}^2$  correctly guesses  $c$ 's label (i.e.,  $\ell_{\text{guess}} = \ell_c$ ) in round  $t$  or earlier. Moreover  $A_{\text{PoS}_o}^2$  makes at most  $q_2^H$  queries to  $H_*$ . We say  $\Lambda_{\text{PoS}_o}(\mathcal{G}, V_C, w)$  is  $(m, t, \epsilon, p_H, q_H)$ -hard if no such  $A_{\text{PoS}_o}$  exists.

## 7.2 $\Phi$ Hardness Implies $\Lambda_{\text{PoSo}}$ Hardness

Before we show that lower bounds in the pebbling game  $\Phi$  translate to lower bounds on the labelling game  $\Lambda_{\text{PoSo}}$ , let us first mention the other (trivial) direction.

Any pebbling strategy  $A_\Phi = \{A_\Phi^1, A_\Phi^2\}$  can be transformed into a labelling strategy  $A_{\text{PoSo}} = \{A_{\text{PoSo}}^1, A_{\text{PoSo}}^2\}$  which has the same parallel time complexity and which uses  $w$  bits of space in its initial state  $S_0$  for pebble in the initial state  $P_0$ . The idea is to simply have  $A_{\text{PoSo}}$  mimic  $A_\Phi$ 's strategy, computing a label whenever  $A_\Phi$  places a pebble.

► **Proposition 6** ((trivial) hardness of  $\Lambda_{\text{PoSo}}$  implies hardness of  $\Phi$ ). *If an  $A_\Phi$  exists that  $(s, t, \epsilon)$ -wins the pebbling game  $\Phi(\mathcal{G} = (V, E), V_C)$ , then an  $A_{\text{PoSo}}$  exists which  $(m, t, \epsilon, q_2^H)$ -wins the  $\Lambda_{\text{PoSo}}(\mathcal{G}, V_C, w)$  labelling game for any  $w$ ,  $q_2^H = |V|$  and*

$$m = s \cdot w$$

**Proof.** By Remark 6.1 we can assume  $A_\Phi^2$  is a “greedy” adversary who never deletes pebbles, and thus puts at most  $|V|$  pebbles on  $\mathcal{G}$  during the entire game. If  $A_\Phi^1$  outputs an initial pebbling  $P_0$ , then  $A_{\text{PoSo}}^1$  will output an initial state that contains all the labels of the pebbles in  $P_0$

$$S_0 = \{\ell_v : v \in P_0\}.$$

Note that  $|S_0| = w \cdot |P_0| \leq w \cdot s$  as claimed.  $A_{\text{PoSo}}^2$  will also be greedy, i.e., store all the labels it ever computes. In step  $i$ , when  $A_\Phi^2$  puts fresh pebbles on  $P_i \setminus P_{i-1}$ ,  $A_{\text{PoSo}}^2$  makes a parallel query to  $H_*$  to compute all the new labels  $\{\ell_v : v \in P_i \setminus P_{i-1}\}$ . In the round where  $A_\Phi^2$  puts a pebble on  $c$ ,  $A_{\text{PoSo}}^2$  can compute and output  $\ell_{\text{guess}} = \ell_c$ . ◀

Proving the other direction – that pebbling lower bounds imply lower bounds on the labelling game – is more challenging.

► **Theorem 7** (hardness of  $\Phi$  implies hardness of  $\Lambda_{\text{PoSo}}$ ). *For any  $\alpha > 0$ , if the pebbling game  $\Phi(\mathcal{G}, V_C)$  is  $(s, t, \epsilon)$ -hard, then the labelling game  $\Lambda_{\text{PoSo}}(\mathcal{G}, V_C, w)$  is  $(m, t, \epsilon, 2^{-\alpha}, q_2^H)$ -hard where*

$$m \geq s \cdot (w - 2(\log N + \log q_2^H)) - \alpha$$

Before we get to proof of this theorem, let us state what security it implies for  $\text{PoSo}$  using the hardness of  $\Phi$  as stated in Lemma 4.

► **Corollary 8** (of Thm. 7 and Lem. 4). *For  $\mathcal{G}_{4N}^{\epsilon'}$ ,  $V_C$  as in Lemma 4, and any  $c_e \in [0, 1]$ ,*

$$\Lambda_{\text{PoSo}}(\mathcal{G}_{4N}^{\epsilon'}, V_C, w) \text{ is } (m, t, \epsilon, 2^{-\alpha}, q_2^H)\text{-hard}$$

$$\text{with } m = N \cdot c_e \cdot (w - 2(\log N + \log q_2^H)) - \alpha, \quad t = N, \quad \epsilon = c_e + 4\epsilon'$$

Let us observe that the hardness as stated is basically optimal. For slightly larger  $m = N \cdot c_e$  (i.e., if we ignore the additive log terms), it means an adversary dedicating  $N \cdot (1 - 4\epsilon' - \Delta) \cdot w$  (instead  $N \cdot w$ ) space after initialization will fail to answer a  $\Delta$  fraction of the challenges in parallel time  $< N$ . Note that in  $4N = |V|$  sequential time every challenge can be answered with no storage at all by recomputing the entire labelling. As always, by challenging this adversary on  $O(1/\Delta)$  queries in parallel we can amplify the probability of the adversary failing to answer fast arbitrary close to 1.

**Proof of Theorem 7.** To prove the theorem we assume an adversary  $A_{PoS_0} = (A_{PoS_0}^1, A_{PoS_0}^2)$  exists who  $(m, t, \epsilon, 2^{-\alpha}, q_2^H)$ -wins the labelling game. Let  $\mathcal{H}, \Pr[H_* \in \mathcal{H}] \geq 2^{-\alpha}$  be the subset of  $H_*$  for which  $A_{PoS_0}$  can win the labelling game like that, i.e., using an initial state of  $\leq m$  bits, in  $\leq t$  rounds, and for an  $\geq \epsilon$  fraction of challenges where  $A_{PoS_0}^2$  makes  $\leq q_2^H$  queries to  $H_*$  (cf. Definition 5).

We will consider the random experiment where for a given  $H_* \in \mathcal{H}$ , we first run  $A_{PoS_0}^1$  to get  $S_0$ , and then we run  $A_{PoS_0}^2$  on all challenges in parallel. This will define a set  $F$  of “fresh” labels, which are labels that occur during the execution *before* they have been actually computed (and thus intuitively must somehow have been stored in the initial state  $S_0$ ). We then prove two claims.

The first shows how the above execution translates into a strategy to  $(|F|, t, \epsilon)$ -win the pebbling game, as this game is  $(s, t, \epsilon)$ -hard, we have  $|F| \geq s$ . The second claim shows how to compress  $H_*$  by almost  $|F| \cdot w$  bits when given the initial state  $S_0$ . As most functions are incompressible, we get  $m = \|S_0\| \gtrsim s \cdot w$ . We now give the detailed proof.

As outlined above, consider any  $H_* \in \mathcal{H}$ , and let  $S_0 \leftarrow A_{PoS_0}^1$  be the initial state. Let  $V'_C \subseteq V_C$  be the set of challenges which  $A_{PoS_0}^2(S_0, \cdot)$  answers correctly in  $t$  rounds or less, as  $H_* \in \mathcal{H}$  we have  $|V'_C| \geq \epsilon|V_C|$ . In the proof we'll consider two algorithms

$A_{PoS_0}^{\parallel}$  runs  $A_{PoS_0}^2$  in parallel for all possible challenges  $c \in V_C$ . Concretely,  $A_{PoS_0}^{\parallel}$  invokes  $|V_C|$  instances of  $A_{PoS_0}^2(S_0, c)$ , one for every challenge  $c \in V_C$ . In each round,  $A_{PoS_0}^{\parallel}$  collects the queries made by all the instances of  $A_{PoS_0}^2$  that have not yet terminated, then makes one parallel query to  $H_*$  containing all the collected queries, and forwards the corresponding answers to the  $A_{PoS_0}^2$  instances. We let  $A_{PoS_0}^{\parallel}$  run for  $t$  rounds, and then stop.

$\mathcal{L}_G$  computes the labels  $\ell_1, \ell_2, \dots, \ell_{|V|}$  in topological order, making sequential queries to  $H_*$ . We refer to a query that correctly computes a label as in eq.(8), i.e., a query of the form

$$\ell_i = H_*(i, \vec{\ell}_{\text{parents}(i)})$$

as a **real query**. For  $i \in V$ , we say  $i$  is **fresh** if in some round  $A_{PoS_0}^{\parallel}$  uses a label  $\ell_i$  as (part of an) input to a query or the thread  $A_{PoS_0}^2(S_0, i)$  outputs  $\ell_i$  as its guess  $\ell_{\text{guess}} = \ell_i$  (note that then  $i \in V'_C$ ) before this label  $\ell_i$  was received as output of a real query. Let  $F \subseteq V$  denote the (indices of) the fresh labels. Thus,  $\{\ell_i\}_{i \in F}$  are all the labels that appear during  $A_{PoS_0}^{\parallel}$ 's execution before they have been computed, i.e., received as output on a real query.

► **Claim 9.** *There is an adversary  $A_{\Phi}$  that  $(s', t, \epsilon)$ -wins the pebbling game  $\Phi(\mathcal{G}, V_C)$  with  $s' = |F|$  initial pebbles (thus  $|F| \geq s$ ).*

**Proof of Claim.** Consider an  $A_{\Phi}^1$  which choses an initial pebbling  $P_0 = F$ . Then  $A_{\Phi}^2$  in round  $i$  puts a pebble on  $v$  if  $A_{PoS_0}^{\parallel}$  received  $\ell_v$  as output of a real query in round  $i$ . By construction this is a valid parallel black pebbling.

We claim that this  $A_{\Phi}^2$  puts a pebble on every node in  $V'_C$  in  $t$  steps or less, and thus  $(s, t, \epsilon)$ -wins  $\Phi(\mathcal{G}, V_C)$ . To see this, consider any  $c \in V'_C$ . If  $c \in F = P_0$  it's pebbled already in round 1. Otherwise, if  $c \in V'_C \setminus F$ , the label  $\ell_{\text{guess}} = \ell_c$  output by the thread  $A_{PoS_0}^2(S_0, c)$  was not fresh, and thus must have been received as output of a real query in some round  $j \leq t$ . By construction this  $A_{\Phi}^2$  will have put a pebble on  $c$  in round no later than  $j$ . ◀

Now that we have shown  $|F| \geq s$ , the next step is to lower bound  $\|S_0\|$ , the bitlength of the initial state, in terms of  $|F|$ . For this, we show how to compress the function table of  $H_*$  given

$S_0$  by almost by almost  $|F| \cdot w$  bits. Using the fact that a random oracle is incompressible (cf. Fact 1 in §3), we'll then derive a lower bound  $\|S_0\| \gtrsim |F| \cdot w$ . Let

$$[H_*] \in \{0, 1\}^{(2^{\ell+1}-1) \times w}$$

denote the function table of  $H_* : \{0, 1\}^{\leq \ell} \rightarrow \{0, 1\}^w$ .

► **Claim 10.** *There exists an encoding (enc, dec) which correctly decodes an  $2^{-\alpha}$  fraction of the tables*

$$\Pr_{H_*}[\text{dec}(\text{enc}([H_*])) = [H_*]] \geq 2^{-\alpha}$$

and the length of the encoding is  $\|\text{enc}([H_*], S_0)\| \leq \|[H_*]\| + \|S_0\| - |F| \cdot (w - 2(\log N + \log q_2^H))$ .

Before we prove this claim, let us observe this implies the statement of the theorem by using Fact 1, which implies

$$\|S_0\| \geq |F| \cdot (w - 2(\log N + \log q_2^H)) - \alpha.$$

**Proof of Claim.** The encoding enc/dec will correctly decode all the  $[H_*]$  which are in  $\mathcal{H}$ . For this,  $\text{enc}([H_*])$  first determines if  $H_* \in \mathcal{H}$ , and if this is not the case outputs whatever (say the bit 0). Let  $B$  denote the following computation: we first invoke  $A_{\text{PoS}_0}^{\parallel}(S_0, V_C)$  followed by  $L_G$ , we'll denote with  $q \leq N \cdot (q_2^H + 1)$  the number of distinct  $H_*$  queries made during  $B$ 's execution (at most  $q_2^H$  for each invocation of the  $N$  invocations of  $A_{\text{PoS}_0}^{\parallel}$  and  $N$  more for  $L_G$ ).

Let the list  $\vec{c}$  contain all the outputs of  $H_*$  queries made during  $B$ . The outputs in  $\vec{c}$  are stored in the order the queries were made, and if a query is repeated, the output is not stored. Let  $\bar{\vec{c}}$  denote the function table of  $H_*$  with the  $|\vec{c}|$   $w$ -bit entries that are in  $\vec{c}$  removed. Note that given  $\vec{c}, \bar{\vec{c}}, S_0, V_{C'}$  we can recover  $[H_*]$  by running  $B$  using  $\vec{c}$  to answer all the oracle queries. After this, we have learned all the inputs corresponding to the outputs stored in  $\vec{c}$ , and thus know which queries were deleted from  $[H_*]$  to get  $\bar{\vec{c}}$ . Thus now we can recover all of  $[H_*]$ . We haven't compressed anything yet (as  $\|\vec{c}\| + \|\bar{\vec{c}}\| = \|[H_*]\|$ , or as all elements in those sets and the table are  $w$  bit strings, equivalently  $|\vec{c}| + |\bar{\vec{c}}| = \|[H_*]\|$ ). Next we'll show how to compress  $\vec{c}$  into a smaller  $\vec{c}_F$  which, with some short extra information  $\vec{b}_F$ , will suffice to answer all  $H_*$  queries made during  $B$  correctly.

Recall that  $F \subseteq V$  are the fresh queries. Consider  $i \in F$ , at some point during the evaluation of  $B$  the real query  $\ell_i = H_*(i, \vec{\ell}_{\text{parents}(i)})$  is made (the only reason we invoke  $L_G$  as part of  $B$  is to ensure this query is made at some point). As  $i \in F$ , at the point where this query is made for the first time, we have already observed the value  $\ell_i$  as part of some query input. Let  $\vec{c}_F$  denote  $\vec{c}$ , but with the  $|F|$  entries corresponding to the real queries of  $i \in F$  deleted. With this  $\vec{c}_F$  we can answer all of  $B$ 's queries if we're given some extra information which, for every  $i \in F$ , tells as at which point during the execution of  $B$  we observe  $\ell_i$ , and where the corresponding real query is made. This extra information requires at most  $2 \log q$  bits for every  $i \in F$ , let  $\vec{b}_F$  denote a string encoding this information, we now define the encoding as

$$\text{enc}([H_*]) = (S_0, \vec{c}_F, \vec{b}_F, \bar{\vec{c}})$$

The decoding  $\text{dec}(S_0, \vec{c}_F, \vec{b}_F, \bar{\vec{c}})$  reconstructs  $[H_*]$  as outlined above. As

$$\begin{aligned} \|\vec{c}_F\| + \|\bar{\vec{c}}\| &= \|[H_*]\| - w \cdot |F| \\ \|\vec{b}_F\| &\leq |F| \cdot 2 \log q \leq |F| \cdot 2(\log N + \log q) \end{aligned}$$

the encoding length is  $\|\text{enc}([H_*], S_0)\| \leq \|S_0\| + \|[H_*]\| - |F| \cdot (w - 2(\log N + \log q))$  as claimed. ◀



## 8 PoS Security of PoCS $_\phi$ and PoR

In this section we extend the result from the previous section, and show that hardness of  $\Phi$  implies hardness of games  $\Lambda_{\text{PoCS}_\phi}$  and  $\Lambda_{\text{PoR}}$ , which capture the PoS security of our constructions PoCS $_\phi$  and PoR. We start with defining the games

### 8.1 The Labelling Games $\Lambda_{\text{PoCS}_\phi}$ and $\Lambda_{\text{PoR}}$

Let  $\text{PoXX} \in \{\text{PoCS}_\phi, \text{PoR}\}$ . The game is parameterized by a DAG  $\mathcal{G} = (V, E)$ , a subset  $V_C \subseteq V$  of  $|V_C| = N$  challenge nodes and a block size  $w$ . Moreover a function  $H_* : \{0, 1\}^{\leq \iota} \rightarrow \{0, 1\}^w$ ,  $\iota = (\delta + 2) \cdot w$ . The game is played by an adversary  $A_{\text{PoXX}} = \{A_{\text{PoXX}}^1, A_{\text{PoXX}}^2\}$ .

**initialization:**  $A_{\text{PoXX}}^1$  is given oracle access to  $H_*$ . It can choose any data  $\vec{d} = \{d_i\}_{i \in V_C}$ ,  $d_i \in \{0, 1\}^w$ , which defines labels  $\vec{\ell}$  to store as in eq.(3) and eq.(4), but using  $H_*$  instead  $H_X$ . Recall that for this we first compute  $(\phi_{\vec{d}}, \phi_{\vec{d}}^+) := \text{commit}^{H_*}(\vec{d})$ , now let  $H_{*, \phi_{\vec{d}}}$  be the function  $H_{*, \phi_{\vec{d}}}(\cdot) \equiv H_*(\phi_{\vec{d}}, \cdot)$ , and then compute  $\vec{\ell}$  as

$$\text{(if PoXX = PoR)} \quad \vec{\ell} = \{\ell_i\}_{i \in V_C} \text{ where } \ell_i = \begin{cases} H_{*, \phi_{\vec{d}}}(i, \vec{\ell}_{\text{parents}(i)}) & \text{if } i \in V \setminus V_C \\ H_{*, \phi_{\vec{d}}}(i, \vec{\ell}_{\text{parents}(i)}) \oplus d_i & \text{if } i \in V_C \end{cases}$$

$$\text{(if PoXX = PoCS}_\phi) \quad \vec{\ell} = \{\ell'_i\}_{i \in V_C} \text{ where } \forall i \in V : \ell_i = H_{*, \phi_{\vec{d}}}(i, \vec{\ell}_{\text{parents}(i)}) \\ \forall i \in V_C : \ell'_i = \ell_i \oplus d_i$$

$A_{\text{PoXX}}^1$  outputs a string (the initial state)  $S_0$  of length  $\|S_0\| = m$  bits.

**execution:** A random challenge node  $c \leftarrow V_C$  is chosen.

$A_{\text{PoXX}}^2$  gets as input  $S_0$  and challenge  $c$ . It then proceeds in rounds starting with  $i = 1$ : In round  $i$ ,  $A_{\text{PoXX}}$  gets as input its state  $S_{i-1}$ . It can either decide to stop the game by outputting a single guess  $\ell_{\text{guess}}$  (for  $\ell_c$  in PoR or  $\ell'_c$  in PoCS $_\phi$ ), or it can make one parallel oracle query: on query  $(x_1, \dots, x_{q_i})$  it receives  $(y_1, \dots, y_{q_i})$  where  $y_i = H_*(x_i)$ . It can do any amount of computation before and after this query, and at the end of the round output its state  $S_i$  for the next round.

► **Definition 11** (hardness of the games  $\Lambda_{\text{PoXX}} \in \{\Lambda_{\text{PoR}}, \Lambda_{\text{PoCS}_\phi}\}$ ). For  $m, t, q_1^H, q_2^H, w \in \mathbb{N}$  and  $p_H, \epsilon \in [0, 1]$ , we say  $A_{\text{PoXX}} = (A_{\text{PoXX}}^1, A_{\text{PoXX}}^2)$  does  $(m, t, \epsilon, p_H, q_1^H, q_2^H)$ -win the labelling game  $\Lambda_{\text{PoXX}}(\mathcal{G}, V_C, w)$  (as defined above) if for all but a  $p_H$  fraction of  $H_*$  the following holds: for at most an  $\epsilon$  fraction of challenges  $c$ ,  $A_{\text{PoXX}}^2$  correctly guesses  $c$ 's label (i.e.,  $\ell_{\text{guess}} = \ell_c$ ) in round  $t$  or earlier. Moreover  $A_{\text{PoXX}}^1$  and  $A_{\text{PoXX}}^2$  make at most  $q_1^H$  and  $q_2^H$  queries to  $H_*$ , respectively. We say  $\Lambda_{\text{PoXX}}(\mathcal{G}, V_C, w)$  is  $(m, t, \epsilon, p_H, q_1^H, q_2^H)$ -hard if no such  $A_{\text{PoXX}}$  exists.

### 8.2 $\Phi$ Hardness Implies $\Lambda_{\text{PoR}}$ and $\Lambda_{\text{PoCS}_\phi}$ Hardness

► **Theorem 12** (hardness of  $\Phi$  implies hardness of  $\Lambda_{\text{PoR}} \& \Lambda_{\text{PoCS}_\phi}$ ). For any  $\alpha > 0$ , if the pebbling game  $\Phi(\mathcal{G}, V_C)$  is  $(s, t, \epsilon)$ -hard, then the labelling game  $\Lambda_{\text{PoR}}(\mathcal{G}, V_C, w)$  and also the labelling game  $\Lambda_{\text{PoCS}_\phi}(\mathcal{G}, V_C, w)$  is  $(m, t, \epsilon, 2^{-\alpha} + q_1^H/2^w, q_1^H, q_2^H)$ -hard where

$$m \geq s \cdot (w - 2(\log N + \log q_2^H)) - \alpha$$

Before we get to proof of this theorem, let us state what security it implies for PoR and PoCS $_\phi$  using the hardness of  $\Phi$  as stated in Lemma 4.

► **Corollary 13** (of Thm. 7 and Lem. 4). For  $\mathcal{G}_{4N}^{\epsilon'}$ ,  $V_C$  as in Lemma 4, and any  $c_e \in [0, 1]$ ,

$$\Lambda_{\text{PoCS}_\phi}(\mathcal{G}_{4N}^{\epsilon'}, V_C, w) \text{ and } \Lambda_{\text{PoR}}(\mathcal{G}_{4N}^{\epsilon'}, V_C, w) \text{ are } (m, t, \epsilon, 2^{-\alpha}, q_2^H)\text{-hard}$$

with  $m = N \cdot c_e \cdot (w - 2(\log N + \log q_2^H)) - \alpha$ ,  $t = N$ ,  $\epsilon = c_e + 4\epsilon'$

**Proof.** We assume the reader is familiar with the proof of Theorem 7, as we will only explain how that proof needs to be adapted.

The proof of Theorem 7 goes through almost unchanged for  $\text{PoXX} \in \{\text{PoR}, \text{PoCS}_\phi\}$  instead of  $\text{PoS}_o$ , the point where it fails is when we need to compress fresh labels. In the proof of Theorem 7 every fresh label  $\ell_i, i \in F$  allowed us to compress one element of  $H_*$ . Now the situation is seemingly more complicated. For concreteness, let's consider  $\text{PoR}$ . Now even if the encoding  $\text{enc}$  observes a fresh label  $\ell_i$  when invoking  $A_{\text{PoR}}^{\parallel}$  (which is defined analogous to  $A_{\text{PoS}_o}^{\parallel}$  in the proof of Theorem 7), it's not clear how to compress one entry of  $H_{*, \phi_{\vec{d}}}$ 's function table as now

$$\ell_i = H_{*, \phi_{\vec{d}}}(i, \vec{\ell}_{\text{parents}(i)}) \oplus d_{\vec{d}, i}$$

only provides an output that is blinded with  $d_{\vec{d}, i}$ . If we could make sure the encoding and decoding  $\text{enc}/\text{dec}$  knew the  $\vec{d}$ , this problem would disappear. We fix this problem as follows. We define  $A_{\text{PoR}}^{\parallel}$  analogous to  $A_{\text{PoS}_o}^{\parallel}$ , i.e., it runs  $A_{\text{PoR}}^2(S_0, c)$  on all challenges  $c \in V_C$  in parallel. But additionally, at the very beginning (before invoking the  $A_{\text{PoR}}^2$ 's), it invokes  $A_{\text{PoR}}^1$ , but only runs it to the point where the commitment  $\phi_{\vec{d}}$  is received as an output of  $H_*$  (recall we assume  $A_{\text{PoR}}^1$  follows the protocol, so this commitment must be computed at some point). This way  $\text{enc}/\text{dec}$ , we invoke  $A_{\text{PoR}}^{\parallel}$ , learn the entire  $\vec{d}$ , as it can be extracted from the  $H_*$  queries leading to  $\phi_{\vec{d}}$ . At the same time  $A_{\text{PoR}}^1$  will almost certainly not have made any  $H_{*, \phi_{\vec{d}}}(\cdot) = H_*(\phi_{\vec{d}}, \cdot)$  queries as  $\phi_{\vec{d}}$  is uniform and we stop executing  $A_{\text{PoR}}^1$  once  $\phi_{\vec{d}}$  is received. This is necessary, so a label that was fresh without running  $A_{\text{PoR}}^1$  first, will still be fresh if we do run  $A_{\text{PoR}}^1$ . The above argument works as long as  $A_{\text{PoR}}^1$  doesn't find a collision in  $H_*$  (otherwise we can't extract a unique  $\vec{d}$ ). For this reason in the theorem security holds only for a slightly smaller  $p_H = 2^{-\alpha} + q_1^{H^2}/2^w$  fraction of the  $H_*$  than the  $p_H = 2^{-\alpha}$  fraction we got for the  $\Lambda_{\text{PoS}_o}$  game in Theorem 7. ◀

---

## References

- 1 Chia Network. <https://chia.network/>, 2017.
- 2 Burstcoin. URL: <http://burstcoin.info>.
- 3 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.
- 4 Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical Graphs for Optimal Side-Channel Resistant Memory-Hard Functions. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 1001–1017. ACM Press, October / November 2017.
- 5 Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-Robust Graphs and Their Cumulative Memory Complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 3–32. Springer, Heidelberg, April / May 2017.
- 6 Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained Space Complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume

- 10821 of *LNCS*, pages 99–130. Springer, Heidelberg, April / May 2018. doi:10.1007/978-3-319-78375-8\_4.
- 7 Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Script Is Maximally Memory-Hard. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 33–62. Springer, Heidelberg, April / May 2017.
  - 8 Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of Useful Work. *IACR Cryptology ePrint Archive*, 2017:203, 2017.
  - 9 Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In David B. Shmoys, editor, *46th ACM STOC*, pages 857–866. ACM Press, May / June 2014.
  - 10 Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. Catalytic Space: Non-determinism and Hierarchy. In *STACS*, volume 47 of *LIPICs*, pages 24:1–24:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
  - 11 Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the Instability of Bitcoin Without the Block Reward. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 154–167. ACM Press, October 2016.
  - 12 Phil Daian, Rafael Pass, and Elaine Shi. Snow White: Provably Secure Proofs of Stake. *Cryptology ePrint Archive*, Report 2016/919, 2016. URL: <http://eprint.iacr.org/2016/919>.
  - 13 Anindya De, Luca Trevisan, and Madhur Tulsiani. Time Space Tradeoffs for Attacks against One-Way Functions and PRGs. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 649–665. Springer, Heidelberg, August 2010.
  - 14 Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing Cracks in the Concrete: Random Oracles with Auxiliary Input, Revisited. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 473–495. Springer, Heidelberg, April / May 2017.
  - 15 Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and Proofs of Work. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 37–54. Springer, Heidelberg, August 2005.
  - 16 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. doi:10.1007/978-3-662-48000-7\_29.
  - 17 Paul Erdős, Ronald L. Graham, and Endre Szemerédi. On Sparse Graphs with Dense Long Paths. Technical report, Stanford University, Stanford, CA, USA, 1975.
  - 18 Ben Fisch. PoReps: Proofs of Space on Useful Data. *Cryptology ePrint Archive*, Report 2018/678, 2018. URL: <https://eprint.iacr.org/2018/678>.
  - 19 Ben Fisch. Tight Proofs of Space and Replication. *Cryptology ePrint Archive*, Report 2018/702, 2018. URL: <https://eprint.iacr.org/2018/702>.
  - 20 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.
  - 21 Eike Kiltz, Krzysztof Pietrzak, and Mario Szegedy. Digital Signatures with Minimal Overhead from Indifferentiable Random Invertible Functions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 571–588. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40041-4\_31.
  - 22 Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake.



- 23 Protocol Labs. Filecoin: A Decentralized Storage Network. <https://filecoin.io/filecoin.pdf>, 2017.
- 24 Daniel Malinowski and Karol Zebrowski. Disproving the Conjectures from “On the Complexity of Scrypt and Proofs of Space in the Parallel Random Oracle Model”. In *ICITS*, volume 10681 of *Lecture Notes in Computer Science*, pages 26–38. Springer, 2017.
- 25 Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.
- 26 Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing Bitcoin Work for Data Preservation. In *2014 IEEE Symposium on Security and Privacy*, pages 475–490. IEEE Computer Society Press, May 2014. doi:10.1109/SP.2014.37.
- 27 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. URL: <https://eprint.iacr.org/2015/528>.
- 28 Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. *Mathematical systems theory*, 10(1):239–251, 1976–1977.
- 29 Krzysztof Pietrzak. Proofs of Catalytic Space. Cryptology ePrint Archive, Report 2018/194, 2018. , full version of this paper. URL: <https://eprint.iacr.org/2018/194>.
- 30 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. doi:10.1007/978-3-662-53641-4\_11.
- 31 The NXT Community. Nxt Whitepaper. <https://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf>, July 2014.

## **A** Discussion and Motivation

### **A.1** The Quest for a Sustainable Blockchain

PoW based blockchains, most notably Bitcoin, have been criticized as the mining process (required to secure the blockchain) results in a massive energy waste. This is not only problematic ecologically, but also economically, as it requires high rewards for the miners to compensate for this energy loss.<sup>8</sup>

#### **A.1.1** Proofs of Stake

The idea behind “Nakamoto consensus” used in Bitcoin, is to randomly chose a miner to generate the next block, where the probability of any miner to be chosen is proportional to its hashing power. The most investigated idea to replace PoWs in blockchains are “proofs of stake” (PoStake), where the idea is to choose the winner proportional to the fraction of coins they hold. At first, this idea looks promising, but it seems to be difficult to actually realize it in a secure and efficient way. Early ad-hoc implementations of this idea include Peercoin [22] and NXT [31]. More recent proposals come with security proofs in various models [25, 12, 20], but those protocols are fairly complicated, and basically run a byzantine agreement protocol amongst rotating subsets of the miners, thus losing the appealing simplicity of Bitcoin, where a winning miner simply gossips the next block and no other interaction is required.

---

<sup>8</sup> The block reward currently used as main compensation for miners in Bitcoin is decreasing (it’s halved every four years), and thus ultimately will be replaced with only transaction fees, which might create serious problems [11].

### A.1.2 Space as a Resource

After time, space is the best investigated resource in computational complexity, it's thus only natural to try using disk space as a resource for mining. This has the potential to give blockchains which are much more sustainable than PoW based designs, while avoiding at least some of the technical issues that PoStake has. Permacoin [26] requires a miner to dedicate disk space, but it's still a proof of work based design, only now the computation itself (which is a so called "proof of retrievability") requires access to a large disk. A proposal which mostly uses space as resource is Burstcoin [2], this design is poorly document, but it seems to have security and efficiency issues.<sup>9</sup> Another suggestion are "proofs of space" (PoS), which are the topic of this paper and we'll discuss them in more detail below in §A.1.4.

### A.1.3 Useful Proofs

While PoStake aim to avoid wasting significant resources for mining in the first place, another approach to minimize the footprint of mining is to use the resources required to sustain the blockchain for something useful. A intriguing idea is to use the computing power wasted for PoWs for solving actual computational problems, we refer the reader to [8] and the references therein. In this work we also follow this approach and construct "proofs of catalytic space" (PoCS), which are defined like PoS, but where most of the space required by the prover can be used to store useful data.

### A.1.4 Proofs of Space (PoS)

Proofs of space (PoS) [16] are proofs systems that were developed to serve as a replacement for PoW in blockchain designs. The first proposal of a PoS-based blockchain is Spacemint [27], a recent ongoing effort which combines PoS with some type of proofs of sequential work is the Chia network [1]. A PoS [16] is a two stage protocol between a prover  $P$  and a verifier  $V$ . The first phase is an intialization protocol which is run only once, after which  $P$  has initialized its space. Then there's a proof execution phase which typically is run many times, in which an honest prover  $P$  can *efficiently* convince the verifier that dedicates the space. The verifier  $V$  is required to be very efficient during both phases, this means it can be polynomial in some security parameter, but should be almost independent (i.e., depend at most polylogarithmically) on the size  $N$  of the space committed by the prover. The honest prover  $P$  is required to be very efficient during the execution phases. During the initialization  $P$  cannot be very efficient, as it must at the very least overwrite all of the claimed space, but it shouldn't require much more than that.

To date two very different types of PoS have been suggested. PoS-based on hard to pebble graphs [16, 30] and PoS-based on inverting random functions [3], the new proofs systems – for proofs of catalytic space (PoCS) and proofs of replication (PoR) – we propose in this work extend the pebbling-based PoS. We leave it as an open problem to extend

---

<sup>9</sup> [27, Appendix B of the full version] discusses some issues with (our best guess on what is) Burst and the underlying proof system called "proofs of capacity" (PoC). In a nutshell, PoC are rather inefficient as the prover needs to access a constant (albeit small) fraction of the entire space for generating a proof, and verification requires over a Million hashes. As to security, PoC allow for strong time-memory trade-offs (a recent ad-hoc fix <https://www.burst-coin.org/wp-content/uploads/2017/07/The-Burst-Dymaxion-1.00.pdf> claims to address at least the most obvious time-memory attacks outlined in [27]). But most worryingly, the blockchain designs seems to have no mechanisms to address nothing-at-stake issues, which are responsible for the most delicate and complicated issue of any blockchain design not based on proofs of work.

the [3] PoS to PoR and PoCS, this would be very interesting as although the [3] PoS has worse *asymptotic* security than pebbling-based PoS, but it's much more efficient (with proofs of length a few hundred bits, and proof generation and verification requiring just a small constant number of hash queries). Moreover, unlike pebbling-based PoS, this PoS has a non-interactive initialization phase, which makes it easier to use it for a blockchain design where we have no dedicated verifier, and thus the proof must be made non-interactive.<sup>10</sup>

---

<sup>10</sup> Concretely, for subtle security reasons Spacemint [27] (which uses the pebbling-based PoS) requires the miners to commit to the transcript of a challenge response protocol which is run during the initialization phase. This is done by uploading this (short) transcript to the blockchain. The Chia network which are based on [3] will not require any such commitments.