Proofs of Catalytic Space

February 17, 2018

Abstract

Proofs of space (PoS) [DFKP15] 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.

The first contribution of this paper is a **security proof for the PoS from [DFKP15]** 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 introduce and construct **proofs of catalytic space** (PoCS), which are defined like classical PoS, but most of the space required by the prover can at the same time be used to store useful data. Our first construction has almost no overhead (i.e., the useful data is almost as large as the dedicated space), whereas our second construction has a slightly larger overhead, but allows for efficient updates of the data. Our constructions are extensions of the [DFKP15] PoS, and our tight proof for the PoS extends (non-trivially) to the PoCS.

As our last contribution we construct a **proof of replication** (PoR), coming up with such an object has recently been stated as an open problem in the Filecoin paper. Also this construction (and its proof) are extensions of the [DFKP15] PoS.

Contents

1 Introduction **2**

1.1 Proofs of Space (PoS) 2

1.2 An Uncoditional Security Proof for the [DFKP15] PoS 4

1.3 Embedding Useful Data into a PoS 4

2 Comparison With Previous Work **6**

3 Basic Notation and Definitions 9
3.1 Notation . . . 9
3.2 Random Oracles . . . 9
3.3 Commitments . . . 10
3.4 Random Strings are Incompressible . . . 10

4 Overview of Our Modes and Protocols 10
4.1 The Mode $\mathsf{E}_{\mathsf{PoS}_\circ}$ . . . 11
4.2 The Mode $\mathsf{E}_{\mathsf{PoR}}$ . . . 11
4.3 The Mode $\mathsf{E}_{\mathsf{PoCS}_\phi}$ . . . 11
4.4 The Mode $\mathsf{E}_{\mathsf{PoCS}_\mathsf{F}}$ . . . 12
4.5 The Protocols $\mathsf{PoS}_\circ, \mathsf{PoCS}_\phi$ and PoR . . . 13

5 The Main Proof Ideas 15

6 The Graph Pebbling Game Φ and its Hardness 17
6.1 The Pebbling Game $\Phi(\mathcal{G}, V_C)$ . . . 17
6.2 Depth Robust Graphs . . . 18
6.3 $\Phi(\mathcal{G}, V_C)$ is Hard if $\mathcal{G}$ is Depth Robust . . . 18

7 PoS Security of $\mathsf{PoS}_\circ$ 19
7.1 The Labelling Game $\Lambda_{\mathsf{PoS}_\circ}(\mathcal{G}, V_C, w)$ . . . 19
7.2 Φ Hardness Implies $\Lambda_{\mathsf{PoS}_\circ}$ Hardness . . . 20

8 PoS Security of PoCS_ϕ and PoR 24
8.1 The Labelling Games $\Lambda_{\mathsf{PoCS}_\phi}$ and Λ_{PoR} . . . 24
8.2 Φ Hardness Implies Λ_{PoR} and $\Lambda_{\mathsf{PoCS}_\phi}$ Hardness . . . 25

A Discussion and Motivation 29
A.1 The Quest for a Sustainable Blockchain . . . 29
A.2 Proofs of Space (PoS) . . . 30

B PoCS_F, a PoCS with Efficient Updates 31
B.1 The Protocol PoCS_F . . . 31
B.2 The Labelling Game $\Lambda_{\mathsf{PoCS}_\mathsf{F}}(\mathcal{G}, V_C, w, \kappa)$ . . . 32
B.3 Φ Hardness Implies $\Lambda_{\mathsf{PoCS}_\mathsf{F}}$ Hardness . . . 33

1 Introduction

1.1 Proofs of Space (PoS)

A proof of space (PoS) [DFKP15, RD16, AAC+17] 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 [PPK+15] and the chia network [chi17].

Some more discussion on sustainable blockchains and PoS which is not crucial for following this work is given in the Appendices §A.1 and §A.2.

The core of the pebbling-based PoS [DFKP15, RD16] is a mode of operation $\mathsf{E}_{\mathsf{PoS}_\circ}$ 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 [DFKP15, RD16] mostly differ in what type of graphs are used. The only input $\mathsf{E}_{\mathsf{PoS}_\circ}$ takes is a short statement χ which is used to sample a hash function H_χ (modelled as random oracle in all our proofs), and it outputs a large file $\boldsymbol{\ell} = \{\ell_i\}_{i \in V_C}$ which P must store. P sends a commitment $\phi_{\boldsymbol{\ell}}$ to $\boldsymbol{\ell}$ 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 $\ell_i \in \boldsymbol{\ell}$ of this file. If such audits happen sufficiently often, the rational thing for P to do is to store $\boldsymbol{\ell}$, and not recompute labels as they are requested.

The high level proof structure of this PoS, denoted $\mathsf{PoS}_\circ$, is illustrated in Figure 1, the underlying mode of operation, denoted $\mathsf{E}_{\mathsf{PoS}_\circ}$, is illustrated in Figure 2.

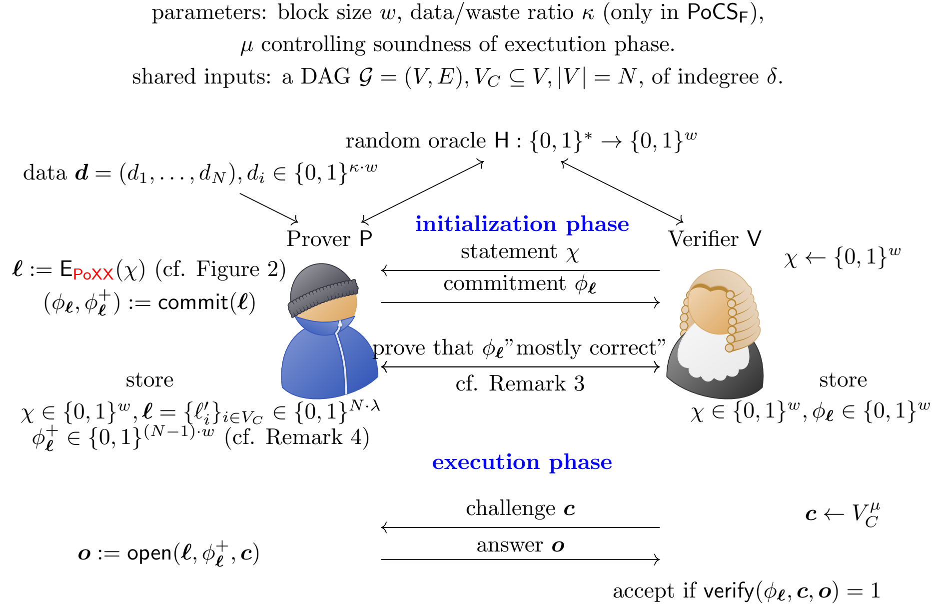


Figure 1: Illustration of protocol structure of the proof of space $\mathsf{PoS}_\circ$, our proofs of catalytic space $\mathsf{PoCS_F}, \mathsf{PoCS}_\phi$ and the proof of replication PoR (replace PoXX in the figure with any of those). In $\mathsf{PoCS_F}$ κ is a parameter, in $\mathsf{PoCS}_\phi, \mathsf{PoR}$ set $\kappa = 1$ and for $\mathsf{PoS}_\circ$ set $\kappa = 0$ (i.e., $\boldsymbol{d}$ is empty) in the figure. The label size λ is $w(\kappa + 1)$ in $\mathsf{PoCS_F}$ and w in $\mathsf{PoS}_\circ, \mathsf{PoR}$ and PoCS_ϕ.

1.2 An Uncoditional Security Proof for the [DFKP15] PoS

Informally, the security we want from a PoS is as follows: if a malicious prover $\tilde{\mathsf{P}}$ 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 $\tilde{\mathsf{P}}$ can always pass the audit by simply recomputing the entire $\boldsymbol{\ell}$ right before the audit, so the best we can hope for is that passing the audit is almost as expensive for $\tilde{\mathsf{P}}$ as it is to compute the entire $\boldsymbol{\ell}$.

The first contribution in this paper is a security proof that shows $\mathsf{PoS}_\circ$ is a secure PoS in the random oracle model (Corollary 11 in §7.2). The existing proof from [DFKP15] only showed security against restricted adversaries who store a subset of the data $\boldsymbol{\ell}$ 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 $\mathsf{E}_{\mathsf{PoS}_\circ}$ 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 blocks in the file $\boldsymbol{\ell}$ the prover is committed to). Thus, if say $\alpha = 2\epsilon$, the prover fails on an ϵ 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 N,[1] 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 ϵ (i.e., the constant grows as ϵ goes to 0).

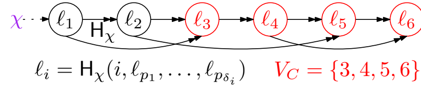
1.3 Embedding Useful Data into a PoS

The file $\boldsymbol{\ell} := \mathsf{E}_{\mathsf{PoS}_\circ}(\chi)$ 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".

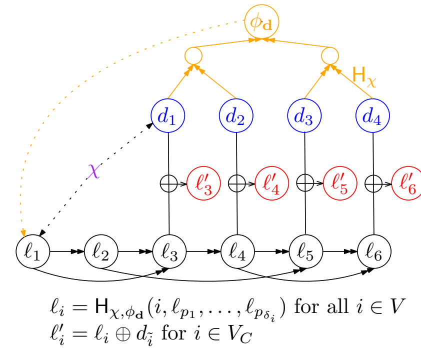
In this paper we investigate the setting where the space dedicated towards the PoS can at the same time be used to encode some useful data $\boldsymbol{d}$. We identify two applications for such objects, "proofs of replication" (PoR), which were (informally) introduced in the Filecoin paper [Lab17] and "proofs of catalytic space" (PoCS), which we introduce and motivate in this work. The naming of the latter is inspired by "catalytic space computations" [BCK+14, BKLS16], 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

[1] 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.

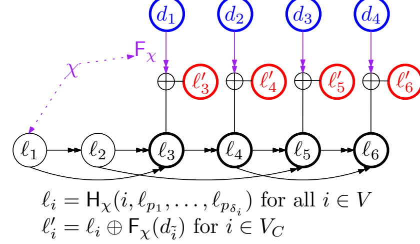
$\mathsf{E}_{\mathsf{PoS}_\circ}$: The **proof of space from [DFKP15]** instantiated with (a toy example of) a depth-robust graph.



$\mathsf{E}_{\mathsf{PoCS}_\phi}$: Our **proof of catalytic space** where the data is committed by a standard Merkle tree commitment $\phi_{\mathbf{d}}$. The hash function for this commitment depends on χ, the hash function for the labelling also on $\phi_{\mathbf{d}}$.



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



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

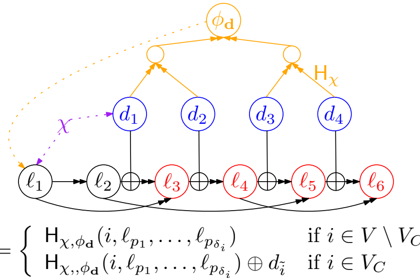


Figure 2: Illustration the graph based modes of operation used in the proof of space $\mathsf{PoS}_\circ$, proof of catalytic space PoCS_ϕ and its efficiently updatable variant PoCS_F and the proof of replication PoR. We use a toy example of a depth robust DAG $\mathcal{G} = (V, E), V = \{1, \ldots, 6\}$ with $V_C = \{3, \ldots, 6\}$ 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 PoCS_F, where the d_i are in $\{0,1\}^{w\cdot\kappa}$ and the $\ell_i\ell'_i, i \in V_C$ are in $\{0,1\}^{w\cdot(\kappa+1)}$.

(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 PoCS_ϕ and PoCS_F. The PoCS_F scheme has a worse "rate" than PoCS_ϕ, by which we mean the ratio $\|\boldsymbol{d}\|/\|\boldsymbol{\ell}\|$ of embedded data vs. dedicated space, but unlike PoCS_ϕ, 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 [DFKP15] PoS $\mathsf{PoS}_\circ$ by replacing its underlying mode $\mathsf{E}_{\mathsf{PoS}_\circ}$ by another mode of operation $\mathsf{E}_{\mathsf{PoXX}} \in \{\mathsf{E}_{\mathsf{PoCS}_\phi}, \mathsf{E}_{\mathsf{PoCS}_\mathsf{F}}, \mathsf{E}_{\mathsf{PoR}}\}$. These modes take as input χ (just like $\mathsf{E}_{\mathsf{PoS}_\circ}$), and additionally some data $\boldsymbol{d} = \{d_i\}_{i \in V_C}$ and output a file $\boldsymbol{\ell} := \mathsf{E}_{\mathsf{PoXX}}(\chi, \boldsymbol{d})$ to be stored. The data $\boldsymbol{d}$ can be recovered from $\boldsymbol{\ell}$ at any time. These four modes are all illustrated in Figure 2.

Properties of PoS, PoCS and PoR. We observe that *a PoCS or a PoR necessarily is also a PoS* as defined in [DFKP15], and we (non trivially) extend our security proof for $\mathsf{PoS}_\circ$ to prove that also the schemes $\mathsf{PoCS}_\phi, \mathsf{PoCS}_\mathsf{F}, \mathsf{PoR}$ constitute PoS (The final bound for $\mathsf{PoS}_\circ, \mathsf{PoCS}_\phi, \mathsf{PoR}$ is stated in Corallary 11 in §8.2, the bound for PoCS_F is in Corollary 19 in §B.3.) 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. We observe that our PoR is not a good PoCS and vice versa, as we'll discuss next.

The most important property we want from a PoCS is that any particular block of data from $\boldsymbol{d} = \{d_i\}_{i \in V_C}$ cannot be recovered too efficiently from $\boldsymbol{\ell}$. The reason is that otherwise the PoCS wouldn't compose: a malicious prover $\tilde{\mathsf{P}}$ could run a PoCS for some statement χ, and at the same time using the embedded data $\boldsymbol{d}$ for a PoCS for other statement χ', 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 $d_i \in \boldsymbol{d}$ should be almost as expensive as recovering the entire $\boldsymbol{d}$ from $\boldsymbol{\ell}$. On the other hand, in a PoR, being able to recover any data block efficiently is actually a nice feature.

In a typical application of a PoR, the data $\boldsymbol{d}$ is not chosen by P but provided by V, together with some replication parameter $r \in \mathbb{N}$ (and statement χ). P will then run the PoR for various statements $\chi_1, \ldots, \chi_r$ (generated from χ), each embedding $\boldsymbol{d}$. Informally, the security property we want is that a prover who later successfully passes the audits must have stored r *redundant* copies of $\boldsymbol{d}$. Note that this implies that PoR is a PoS.

In §4, where we define our modes, we'll explain in more detail how they fail to be good PoCS or PoR. In this work we do not provide formal definitions of the above mentioned "locking" and "replication" property, and it indeed seems non-trivial to come up with the right definitions here. Summing up, in this work we prove that the PoS from [DFKP15] and our three new proof systems that allow for embedding useful data are indeed PoS (which is a necessary property for being a PoR or PoCS), but we do not prove that any of them has the locking or replication property. We only show in §4 that each of our constructions fails to have one or the other. There certainly are more properties one might need from a PoR or PoCS in particular applications that we have not identified, coming up with the right definitions and constructions satisfying them seems like a promising research agenda.

2 Comparison With Previous Work

We somewhat divert from [DFKP15] when formally defining the security as a PoS. In [DFKP15], a PoS is defined to be (N_0, N_1, T)-secure if an adversary who stores a file $\tilde{\boldsymbol{\ell}}$ 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. We will shortly discuss the three instantiations

of the $\mathsf{PoS}_\circ$ 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 δ, as this determines the size of the proofs and also the time to generate and verify proof.

The [DFKP15] Constructions and Proofs. In [DFKP15] two PoS were proposed, the first is

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

and based on a graph with high space pebbling complexity by Paul, Tarjan and Celoni [PTC77], the second uses a rather sophisticated construction combining random bipartite graphs, superconcentrators and depth-robust graphs [EGS75] and is

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

The [RD16] Construction and Proof. Ren and Devadas [RD16] propose a very elegant instantiation of $\mathsf{PoS}_\circ$ using stacked expanders and give a proof for it which in terms of security improves upon both constructions from [DFKP15] (just $|E|$ is asymptotically larger). For any $\alpha \in [0, 0.5]$, their proof implies security (almost)

$$(\alpha \cdot N, (1-\alpha) \cdot N, \infty)\text{-secure with} |V| = N \log N, \delta = 2, |E| = 2N \log N$$

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.

Our Proof. In this work we use the depth-robust graphs from [ABP17] to instantiate $\mathsf{PoS}_\circ$, and also our three new constructions which allow to embed useful data. For any $\epsilon > 0$, we can instantiate it as to get

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

This might not seem terribly impressive, note that unlike [RD16] 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 [DFKP15] which (ignoring constants) has the same security. But as we'll explain next, we improve upon all existing constructions in three crucial points.

1. **Unconditional Proof:** Our proof holds unconditionally (in the random oracle model), whereas [DFKP15, RD16] 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.[2]

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 ϵ 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) [RD16] 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 reinitalize the entire space once the audit starts fast enough using massive parallelism. Compare this to the construction from [RD16], which can be initalized in sequential time $\log(N)$ using parallelism N.

We will not use the formalism from [DFKP15] 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 $\mathsf{E}_{\mathsf{PoS}_\circ}$ is stated in Corollary 11, The PoS security of $\mathsf{E}_{\mathsf{PoCS}_\phi}$ and $\mathsf{E}_{\mathsf{PoR}}$ in Corollary 16 and the PoS security of $\mathsf{E}_{\mathsf{PoCS}_\mathsf{F}}$ in Corallary 19.

[2]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 [ABP17] 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.

3 Basic Notation and Definitions

3.1 Notation

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

We typically use small greek letters $\iota, \delta, \omega, \mu, \nu, \epsilon, \ldots$ 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 ϵ which is in $[0, 1]$. For the security games considered in this paper we use capital greek letters Φ, Λ. The sans-serif font is used for interactive systems like parties $\mathsf{V}, \mathsf{P}, \tilde{\mathsf{P}}, \mathsf{A}$ (modelled as randomized interactive Turing machines), functions $\mathsf{H}, \mathsf{F}, \mathsf{g}, \mathsf{f}$ or algorithms like commitments discussed below (V, P and $\tilde{\mathsf{P}}$ are reserved for an honest verifier, an honest prover, and a potentially malicious prover, respectively). We use bold letters $\boldsymbol{\ell}, \boldsymbol{d}, \boldsymbol{o}, \boldsymbol{c}, \ldots$ 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.,

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

3.2 Random Oracles

Fresh Random Oracles. If H is a fixed random oracle and $z \in \{0,1\}^*$, we denote with H_z the function $\mathsf{H}_z(\cdot) = \mathsf{H}(z, \cdot)$. If z is random and long enough (concretely, the amount of non-uniform advice an adversary has on H is a not too large exponential in $\|z\|$), we can treat H_z as a fresh uniformly random oracle [DGK17]. We do this repeatedly in this work without always explicitly mentioning it.

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 $x_1, \ldots, x_i$ of queries, and it receives the outputs $\mathsf{H}(x_1), \ldots, \mathsf{H}(x_i)$ 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 $\mathsf{PoS}_\circ$, 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 $\mathsf{commit}, \mathsf{open}, \mathsf{verify}$ which use a hash-function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$ 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 commiting 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 $\boldsymbol{x} = (x_1, \ldots, x_m)$ invokes

$$(\phi_{\boldsymbol{x}}, \phi_{\boldsymbol{x}}^+) := \mathsf{commit}^{\mathsf{H}}(\boldsymbol{x})$$

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

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

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

Everyone can verify that $\boldsymbol{o}$ is the correct opening by invoking $\mathsf{verify}^{\mathsf{H}}(\phi_{\boldsymbol{x}}, \boldsymbol{i}, \boldsymbol{o})$ 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 [DTT10])**.** *For any randomized encoding procedure* $\mathsf{enc} : \{0,1\}^r \times \{0,1\}^n \to \{0,1\}^m$ *and decoding procedure* $\mathsf{dec} : \{0,1\}^r \times \{0,1\}^m \to \{0,1\}^n$ *where*

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

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

4 Overview of Our Modes and Protocols

In this section we will formally define the mode $\mathsf{E}_{\mathsf{PoS}_\circ}$ underlying the PoS from [DFKP15] and our new modes $\mathsf{E}_{\mathsf{PoR}}, \mathsf{E}_{\mathsf{PoCS}_\phi}, \mathsf{E}_{\mathsf{PoCS}_\mathsf{F}}$ as illustrated in Figure 2. The actual protocols using those modes as illustrated in Figure 1 will then be defined in §4.5 ($\mathsf{PoS}_\circ, \mathsf{PoR}, \mathsf{PoCS}_\phi$), and §B.1 ($\mathsf{PoCS}_\mathsf{F}$). 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 $\mathsf{E}_{\mathsf{PoR}}$ is

not locking (and thus not suitable as PoCS), whereas $\mathsf{E}_{\mathsf{PoCS}_\phi}, \mathsf{E}_{\mathsf{PoCS}_\mathsf{F}}$ do not imply replication. All the modes are defined over a DAG $\mathcal{G} = (V, E)$, for $i \in V$ we denote with $\mathsf{parents}(i) = \{j : (j,i) \in E\}$ the parents of i, and we define $\boldsymbol{\ell}_{\mathsf{parents}(i)} = \{\ell_j \ : \ j \in \mathsf{parents}(i)\}$.

4.1 The Mode $\mathsf{E}_{\mathsf{PoS}_\circ}$

In the basic PoS the file $\boldsymbol{\ell} := \mathsf{E}_{\mathsf{PoS}_\circ}(\chi)$ 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 χ) the labels of its parents

$$\forall i \in V : \ell_i = \mathsf{H}_\chi(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \quad , \quad \boldsymbol{\ell} \stackrel{\mathsf{def}}{=} \{\ell_i\}_{i \in V_C} \qquad (\mathsf{E}_{\mathsf{PoS}_\circ}) \tag{2}$$

The most obvious way to somehow embed data $\boldsymbol{d} = \{d_i\}_{i \in [N]}$ 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 PoCS_ϕ. Before computing the labels, we first commit to $\boldsymbol{d}$, 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 $\mathsf{E}_{\mathsf{PoR}}$

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

$$(\phi_{\boldsymbol{d}}, \phi^+_{\boldsymbol{d}}) := \mathsf{commit}^{\mathsf{H}_\chi}(\boldsymbol{d}) \ .$$

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

$$\ell_i = \begin{cases} \mathsf{H}_{\chi,\phi_{\boldsymbol{d}}}(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) & \text{if } i \in V \setminus V_C \\ \mathsf{H}_{\chi,\phi_{\boldsymbol{d}}}(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \oplus d_{\tilde{i}} & \text{if } i \in V_C \end{cases} \quad , \quad \boldsymbol{\ell} = \{\ell_i\}_{i \in V_C} \qquad (\mathsf{E}_{\mathsf{PoR}}) \tag{3}$$

4.3 The Mode $\mathsf{E}_{\mathsf{PoCS}_\phi}$

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

$$\forall i \in V \ : \ \ell_i \ = \ \mathsf{H}_{\chi,\phi_{\boldsymbol{d}}}(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \tag{4}$$

$$\forall i \in V_C \ : \ \ell'_i \ = \ \ell_i \oplus d_{\tilde{i}} \quad , \quad \boldsymbol{\ell} = \{\ell'_i\}_{i \in V_C} \qquad (\mathsf{E}_{\mathsf{PoCS}_\phi}) \tag{5}$$

As mentioned above, the fact that the random oracle $\mathsf{H}_{\chi,\phi_{\boldsymbol{d}}}$ used to computed the labels depends on the commitment $\phi_{\boldsymbol{d}}$ to $\boldsymbol{d}$ is crucial, let us sketch why. Assume we'd change $\mathsf{H}_{\chi,\phi_{\boldsymbol{d}}}$ to H_χ in the definition of $\mathsf{E}_{\mathsf{PoCS}_\phi}$. Now a malicious prover (in the protocol PoCS_ϕ to be defined in §4.5) could set the ℓ'_i to be stored to whatever it wants (and thus also store them using low space). Only after choosing the ℓ'_i, it then fixes the data $\boldsymbol{d} = \{d_i\}_{i\in[N]}$ by "equivocating" it, i.e., setting it as $d_{\tilde{i}} := \ell'_i \oplus \ell_i$, so everything is consistent. This malicious behaviour (not using any space) cannot be distinguished from honest behaviour, thus it's not a PoS.[3]

Now let us observe that PoCS_ϕ is not a good PoR, as it doesn't imply replication. A prover who is supposed to compute and store $\boldsymbol{\ell}_i := \mathsf{PoCS}_\phi(\chi_i, \boldsymbol{d})$ for r statements $\chi_1, \ldots, \chi_r$ but the same $\boldsymbol{d}$ can instead store $\boldsymbol{d}$ once in the clear, and for then for each χ_j only store the labels $\{\ell_i\}_{i\in V_C}$ as in eq.(4) instead $\{\ell'_i = \ell_i \oplus d_{\tilde{i}}\}_{i\in V_C}$, i.e., avoid the XORing step of eq.(5). Note that this prover has only stored one copy of $\boldsymbol{d}$, 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 ℓ'_i using its single copy of $\boldsymbol{d}$ as $\ell'_i = \ell_i \oplus d_{\tilde{i}}$. 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 $\boldsymbol{d}$ to the disk.

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

4.4 The Mode $\mathsf{E}_{\mathsf{PoCS}_\mathsf{F}}$

Besides PoCS_ϕ, we propose a second PoCS PoCS_F, which allows for efficient updates. Instead of committing to $\boldsymbol{d}$ and using this commitment to sample the random oracle $\mathsf{H}_{\chi,\phi_{\boldsymbol{d}}}$ for computing the labels as in PoCS_ϕ, in PoCS_F the labels are computed directly using H_χ, i.e., independently of $\boldsymbol{d}$. To prevent the "equivocation" attack outlined above, in PoCS_F the prover samples (using χ) a random invertible function $\mathsf{F}_\chi : \{0,1\}^{(\kappa+1)\cdot w} \to \{0,1\}^{\kappa\cdot w}$ and applies it to the data before XORing it to the label, where κ 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 $(\kappa+1)\cdot w$

[3]In our proofs we will actually assume the prover is honest during initialization, which can be done wlog. as we'll outline in Remark 3 because cheating can be easily detected. This is not the case for the outlined here, as "equivocating" the d_i's is indistinguishable from being honest behaviour.

bits long. Below $\mathsf{H}_\chi : \{0,1\}^{\leq \iota} \to \{0,1\}^{(\kappa+1)\cdot w}$, and $\mathsf{H}_\chi(\cdot)_{|w}$ means we cap the output after w bits.

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

$$\forall i \in V_C \ : \ \ell'_i \quad = \quad \ell_i \oplus \mathsf{F}_\chi(d_{\tilde{i}}) \quad , \quad \boldsymbol{\ell} = \{\ell'_i\}_{i \in V_C} \qquad (\mathsf{E}_{\mathsf{PoCS}_\phi}) \tag{7}$$

A random invertible function F_χ 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 [KPS13].

PoCS_ϕ has a better rate than PoCS_F. In PoCS_ϕ we have rate $\frac{\|\boldsymbol{d}\|}{\|\boldsymbol{\ell}\|} = 1$, so the stored file $\boldsymbol{\ell}$ is as big as the data $\boldsymbol{d}$ that can be recovered from it.[4] The rate of PoCS_F is only $\frac{\|\boldsymbol{d}\|}{\|\boldsymbol{\ell}\|} = \frac{\kappa}{\kappa+1}$. For efficiency reasons the κ can't be too large ($\kappa \approx 10$ is reasonable, then the size of $\boldsymbol{\ell}$ is $\approx 10\%$ larger than $\boldsymbol{d}$).

But unlike PoCS_ϕ, PoCS_F allows for fast updates of the catalytic data: if P wants to change a single data block $d_{\tilde{i}}$ (embedded in $\ell'_i = \ell_i \oplus \mathsf{F}_\chi(d_{\tilde{i}})$) to $d'_{\tilde{i}}$ then it can simply replace the label ℓ'_i with $\ell'_i \oplus \mathsf{F}_\chi(d_{\tilde{i}}) \oplus \mathsf{F}_\chi(d'_{\tilde{i}})$. It then must also update the commitment $(\phi_{\boldsymbol{\ell}}, \phi^+_{\boldsymbol{\ell}})$, but this takes only time $\log(N)$. For this update P actually needs to know the currently embedded data block $d_{\tilde{i}}$. There are natural settings where $\boldsymbol{d}$ is readily available in the clear. For example if the catalytic data $\boldsymbol{d}$ encoded in $\boldsymbol{\ell}$ is used as backup, and a working copy of $\boldsymbol{d}$ is available in the clear. So we think this feature might be useful. As this mode is somewhat different than the other three modes considered, we mostly define and analyze the $\mathsf{PoS}_\circ, \mathsf{PoR}, \mathsf{PoCS}_\phi$ together, but we'll postpone the definition and proofs on PoCS_F to §B.

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

4.5 The Protocols $\mathsf{PoS}_\circ, \mathsf{PoCS}_\phi$ and PoR

The protocols we consider in this work are a generalization of the pebbling-based PoS from [DFKP15] $\mathsf{PoS}_\circ$, where we allow the prover to chose and embed additional data $\boldsymbol{d}$ into the file $\boldsymbol{\ell}$ to be stored. We define $\mathsf{PoS}_\circ, \mathsf{PoCS}_\phi, \mathsf{PoR}$

[4] The prover also must store opening information $\phi^+_{\boldsymbol{\ell}}$ for a Merkle commitment, but as we'll discuss in Remark 4 this is small compared to $\boldsymbol{\ell}$.

together as they are very similar ($\mathsf{PoCS_F}$ is somewhat different, and only defined in §B), we use $\mathsf{PoXX} \in \{\mathsf{PoS}_\circ, \mathsf{PoCS}_\phi, \mathsf{PoR}\}$ as placeholder for any of those constructions.

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

$\mathcal{G}$: A directed acyclic graph $\mathcal{G} = (V, E)$ with max. indegree δ 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 $\mathsf{H} : \{0,1\}^{\leq \iota} \to \{0,1\}^w$ 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).

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

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

$$\boldsymbol{\ell} := \mathsf{E}_{\mathsf{PoXX}}(\chi, \boldsymbol{d})$$

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

Finally P computes the commitment $(\phi_{\boldsymbol{\ell}}, \phi^+_{\boldsymbol{\ell}}) := \mathsf{commit}(\boldsymbol{\ell})$ for all the labels $i \in V_C$, sends the short commitment $\phi_{\boldsymbol{\ell}}$ to V and locally stores the opening information $\phi^+_{\boldsymbol{\ell}}$. This concludes the initialization if we assume P is honest during this phase (we'll discuss the general case in Remark 3).

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

Proof execution. The protocol where $\mathsf{P}(\boldsymbol{\ell}, \chi, \phi^+_{\boldsymbol{\ell}})$ convinces $\mathsf{V}(\chi, \phi_{\boldsymbol{\ell}})$ that it stores $\boldsymbol{\ell}$ is very simple. V samples a few nodes from the challenge set $\boldsymbol{c} = (c_1, \ldots, c_\mu) \subset V_C$ at random, and sends the challenge $\boldsymbol{c}$ to P. P sends openings $\boldsymbol{o} := \mathsf{open}(\boldsymbol{\ell}, \phi^+_{\boldsymbol{\ell}}, \boldsymbol{c})$ to the labels $\{\ell_{\tilde{i}}\}_{i \in \boldsymbol{c}}$ to V, who then accepts iff $\mathsf{verify}(\phi_{\boldsymbol{\ell}}, \boldsymbol{c}, \boldsymbol{o}) = 1$.

Remark 3 (prover is honest during initialization)**.** *For most of the paper we will assume that even a malicious prover $\tilde{\mathsf{P}}$ 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 ν – V challenges $\tilde{\mathsf{P}}$ to open ν labels and their parents to check that they were correctly computed. For this, $\tilde{\mathsf{P}}$ initially sends a commitment to all nodes V, not just V_C, and (except for $\mathsf{PoS}_\circ$) also a commitment to $\boldsymbol{d}$. If $\tilde{\mathsf{P}}$ committed to labels $\{\ell^*_i\}_{i \in V}$ where it cheated on an ϵ fraction of the lables, i.e., for $\mathsf{PoS}_\circ$ this means we have $\ell^*_i \neq \mathsf{H}_\chi(i, \ell^*_{p_1}, \ldots, \ell^*_{p_{\delta_i}})$, then $\tilde{\mathsf{P}}$ will fail to pass*

this check with probability $1 - (1 - \epsilon)^\nu$*.* $\tilde{\mathsf{P}}$ *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 4 (P's space)**.** *The size of the file* $\boldsymbol{\ell}$ *is* $N \cdot w$ *bits, which is basically the same as the* $(N-1) \cdot w$ *bits required to store the opening info* $\phi^+_{\boldsymbol{\ell}}$ *of the Merkle-tree commitment to* $\boldsymbol{\ell}$*: 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* $N/2 + N/4 + \ldots + 2 + 1 = N - 1$ *labels of internal nodes that constitute* $\phi^+_{\boldsymbol{\ell}}$*. But the prover can decide to not store levels* $1 \ldots k$*, 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* $\boldsymbol{\ell}$*). Thus, for say* $k = 5$*, the Merkle tree requires just a* $1/32$ *fraction of the space of* $\boldsymbol{\ell}$*, but requires to hash* 32 *values. In practice that wouldn't be expensive as those leave 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 this opening information* $\phi^+_{\boldsymbol{\ell}}$*.*

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) [ABP17] and sustained space (the number of rounds at which many pebbles were on the graph) [ABP18] 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 [DNW05] 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.[5]

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 [ACP+17], 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 Φ and defined in §6.1) underlying pebbling-based PoS [DFKP15, RD16] 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.[6]

The key observation that allows us to translate pebbling-lower bounds for a "randomized" pebbling game like Φ to lower bounds for the "labelling game" $\Lambda_{\mathsf{PoS}_\circ}$ in the random oracle model (this game is defined in §7.1) is already implicit in [ACP+17], 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

[5]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 requrired 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.

[6]In [DFKP15] some combinatorial conjectures were statet which – if true – would have implied that restricting adversaries like this is without loss of generality. But these conjectures have been beautifully refuted in [MZ17].

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 10 in §7.2) showing that lower bounds on the hardness of the game Φ imply lower bounds on the game $\Lambda_{\mathsf{PoS}_\circ}$, which captures the security of $\mathsf{PoS}_\circ$ 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 15 in §8.2 we extend this proof from the basic $\mathsf{PoS}_\circ$ to the modes PoCS_ϕ and PoR. The problem we face is that now, the values this "too successful" adversary predicts are not just outputs (i.e., labels ℓ_i as in $\mathsf{E}_{\mathsf{PoS}_\circ}$) of the random oracle H, but now they are of the form $\ell_i \oplus d_{\tilde{i}}$, where $d_{\tilde{i}}$ 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 $\mathsf{PoCS}_\phi, \mathsf{PoR}$, 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 $\mathsf{PoS}_\circ$.

Extending the proof to show that our efficiently updatable PoCS PoCS_F is a PoS is much more challenging, and is done in the last section §B of this paper. Here the labels of the "too succesful" adversary predicts are of the form $\ell_i \oplus \mathsf{F}(d_{\tilde{i}})$, 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 $\mathsf{F} : \{0,1\}^{\lambda-w} \to \{0,1\}^\lambda$ with the composition of two randomly sampled functions $\mathsf{g}(\mathsf{f}(\cdot))$, where $\mathsf{f} : \{0,1\}^{\lambda-w} \to \{0,1\}^{w/2}, \mathsf{g} : \{0,1\}^{w/2} \to \{0,1\}^\lambda$, 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 ℓ_i from the labels the adversary predicts, which now are of the form $\ell_i \oplus \mathsf{g}(\mathsf{f}(d_{\tilde{i}}))$, if additionally given only the short $w/2$ bit string $\mathsf{f}(d_{\tilde{i}})$, this is good enough to get compression and again push through an ex-post-facto type proof.

6 The Graph Pebbling Game Φ and its Hardness

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

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 $\mathsf{A}_\Phi = \{\mathsf{A}^1_\Phi, \mathsf{A}^2_\Phi\}$.

initialization: A^1_Φ 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.

A^2_Φ gets as input P_0 and the challenge c. It then proceeds in rounds, starting at round 1.

In round i, A^2_Φ 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 5 (hardness of the game Φ)**.** *For $s, t \in \mathbb{N}, \epsilon \in [0,1]$, we say A_Φ does (s,t,ϵ)-win the pebbling game $\Phi(\mathcal{G}, V_C)$ (as defined above) if the probability (over the choice of c and A_Φ's random coins) that A^2_Φ puts a pebble on c in $t-1$ rounds or less is at most ϵ.*

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

Remark 6 (greedy is best)**.** *We observe that the optimal strategy for A^2_Φ is trivial: the greedy strategy, where in every round A^2_Φ 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 7 (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. [EGS75], and recent work has made them more practical, cf. [ABH17, ABP18] and references therein. Concretely, for any $\epsilon > 0$, [ABP18] constructs a family $\{\mathcal{G}^\epsilon_N\}_{N\in\mathbb{N}}$ of graphs of indegree $O(\log N)$ (here the hidden constant depends on ϵ) 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 (one 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 $\Lambda_{\mathsf{PoS}_\circ}(\mathcal{G}, V_C)$ 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 8 (hardness of Φ with the depth-robust graphs from [ABP18])**.** *For any $N \in \mathbb{N}, \epsilon' > 0$ consider the graph $\mathcal{G}_{4N}^{\epsilon'}$ from [ABP18], 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,ϵ)-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. ■

7 PoS Security of $\mathsf{PoS}_\circ$

In this Section we state and prove our main technical result Theorem 1, which states that hardness of the pebbling game Φ implies hardness of a game $\Lambda_{\mathsf{PoS}_\circ}$ capturing the PoS security of $\mathsf{PoS}_\circ$. We start with defining the the $\Lambda_{\mathsf{PoS}_\circ}$ game

7.1 The Labelling Game $\Lambda_{\mathsf{PoS}_\circ}(\mathcal{G}, V_C, w)$

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 $\mathsf{H}_* : \{0,1\}^{\leq \iota} \to \{0,1\}^w, \iota = (\delta + 2) \cdot w$. Let

$$\forall i \in V \ : \ \ell_i = \mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \tag{8}$$

(note that these are the labels $\mathsf{PoS}_\circ(\chi)$ would compute if $\mathsf{H}_\chi = \mathsf{H}_*$). The game is played by an adversary $\mathsf{A}_{\mathsf{PoS}_\circ} = \{\mathsf{A}^1_{\mathsf{PoS}_\circ}, \mathsf{A}^2_{\mathsf{PoS}_\circ}\}$

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

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

$\mathsf{A}^2_{\mathsf{PoS}_\circ}$ gets as input S_0 and the challenge c. It then proceeds in rounds, starting at round 1.

In round i, $\mathsf{A}_{\mathsf{PoS}_\circ}$ gets as input its state S_{i-1}. It can either decide to stop the game by outputting a single guess ℓ_{guess} (for ℓ_c), or it can make one parallel oracle query: on query $(x_1, \ldots, x_{q_i})$ it receives $(y_1, \ldots, y_{q_i})$ where $y_i = \mathsf{H}_*(x_i)$. 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 9 (hardness of the game $\Lambda_{\mathsf{PoS}_\circ}$)**.** *For $m, t, q^{\mathsf{H}}_2, w, \alpha \in \mathbb{N}$ and $p_{\mathsf{H}}, epsilon \in [0, 1]$, we say $\mathsf{A}_{\mathsf{PoS}_\circ} = (\mathsf{A}^1_{\mathsf{PoS}_\circ}, \mathsf{A}^2_{\mathsf{PoS}_\circ})$ does $(m, t, \epsilon, p, q^{\mathsf{H}}_2)$-win the labelling game $\Lambda_{\mathsf{PoS}_\circ}(\mathcal{G}, C, w)$ (as defined above) if for all but a p_{H} fraction of H_* the following holds: for at most an ϵ fraction of challenges c, $\mathsf{A}^2_{\mathsf{PoS}_\circ}$ correctly guesses c's label (i.e., $\ell_{\mathsf{guess}} = \ell_c$) in round t or earlier. Moreover $\mathsf{A}^2_{\mathsf{PoS}_\circ}$ makes at most q^{H}_2 queries to H_*. We say $\Lambda_{\mathsf{PoS}_\circ}(\mathcal{G}, V_C, w)$ is $(m, t, \epsilon, p_{\mathsf{H}}, q_{\mathsf{H}})$-hard if no such $\mathsf{A}_{\mathsf{PoS}_\circ}$ exists.*

7.2 Φ Hardness Implies $\Lambda_{\mathsf{PoS}_\circ}$ Hardness

Before we show that lower bounds in the pebbling game Φ translate to lower bounds on the labelling game $\Lambda_{\mathsf{PoS}_\circ}$, let us first mention the other (trivial) direction.

Any pebbling strategy $\mathsf{A}_\Phi = \{\mathsf{A}^1_\Phi, \mathsf{A}^2_\Phi\}$ can be transformed into a labelling strategy $\mathsf{A}_{\mathsf{PoS}_\circ} = \{\mathsf{A}^1_{\mathsf{PoS}_\circ}, \mathsf{A}^2_{\mathsf{PoS}_\circ}\}$ 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 $\mathsf{A}_{\mathsf{PoS}_\circ}$ mimic A_Φ's strategy, computing a label whenever A_Φ places a pebble.

Proposition 1 ((trivial) hardness of $\Lambda_{\mathsf{PoS}_\circ}$ implies hardness of Φ)**.** *If an A_Φ exists that (s, t, ϵ)-wins the pebbling game $\Phi(\mathcal{G} = (V, E), V_C)$, then an $\mathsf{A}_{\mathsf{PoS}_\circ}$ exists which $(m, t, \epsilon, q^{\mathsf{H}}_2)$-wins the $\Lambda_{\mathsf{PoS}_\circ}(\mathcal{G}, V_C, w)$ labelling game for any w, $q^{\mathsf{H}}_2 = |V|$ and*

$$m = s \cdot w$$

Proof. By Remark 6 we can assume A^2_Φ is a "greedy" adversary who never deletes pebbles, and thus puts at most $|V|$ pebbles on $\mathcal{G}$ during the entire game. If A^1_Φ outputs an initial pebbling P_0, then $\mathsf{A}^1_{\mathsf{PoS}_\circ}$ 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 $|S_0| = w \cdot |P_0| \le w \cdot s$ as claimed. $\mathsf{A}^2_{\mathsf{PoS}_\circ}$ will also be greedy, i.e., store all the labels it ever computes. In step i, when A^2_Φ puts fresh pebbles on $P_i \setminus P_{i-1}$, $\mathsf{A}^2_{\mathsf{PoS}_\circ}$ 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^2_Φ puts a pebble on c, $\mathsf{A}^2_{\mathsf{PoS}_\circ}$ can compute and output $\ell_{\mathsf{guess}} = \ell_c$. ■

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

Theorem 10 (hardness of Φ implies hardness of $\Lambda_{\mathsf{PoS}_\circ}$)**.** *For any $\alpha > 0$, if the pebbling game $\Phi(\mathcal{G}, V_C)$ is (s, t, ϵ)-hard, then the labelling game $\Lambda_{\mathsf{PoS}_\circ}(\mathcal{G}, V_C, w)$ is $(m, t, \epsilon, 2^{-\alpha}, q_2^{\mathsf{H}})$-hard where*

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

Before we get to proof of this theorem, let us state what security it implies for $\mathsf{PoS}_\circ$ using the hardness of Φ as stated in Lemma 8.

Corollary 11 (of Thm. 10 and Lem. 8)**.** *For $\mathcal{G}_{4N}^{\epsilon'}, V_C$ as in Lemma 8, and any $c_e \in [0, 1]$,*

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

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

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 and adversary dedicating $N \cdot (1 - 4\epsilon' - \Delta) \cdot w$ (instead $N \cdot w$) space after initialization will fail to answer a Δ 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 10. To prove the theorem we assume an adversary $\mathsf{A}_{\mathsf{PoS}_\circ} = (\mathsf{A}^1_{\mathsf{PoS}_\circ}, \mathsf{A}^2_{\mathsf{PoS}_\circ})$ exists who $(m, t, \epsilon, 2^{-\alpha}, q_2^{\mathsf{H}})$-wins the labelling game. Let $\mathcal{H}, \Pr[\mathsf{H}_* \in \mathcal{H}] \geq 2^{-\alpha}$ be the subset of H_* for which $\mathsf{A}_{\mathsf{PoS}_\circ}$ 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 $\mathsf{A}^2_{\mathsf{PoS}_\circ}$ makes $\leq q_2^{\mathsf{H}}$ queries to H_* (cf. Definition 14).

We will consider the random experiment where for a given $\mathsf{H}_* \in \mathcal{H}$, we first run $\mathsf{A}^1_{\mathsf{PoS}_\circ}$ to get S_0, and then we run $\mathsf{A}^2_{\mathsf{PoS}_\circ}$ 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 he the pebbling game, as this game is (s, t, ϵ)-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 $m = \|S_0\| \gtrapprox s \cdot w$. We now give the detailed proof.

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

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

$\mathsf{L}_{\mathcal{G}}$ 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

$$\ell_i = \mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)})$$

as a **real query**. For $i \in V$, we say i is **fresh** if in some round $\mathsf{A}^{\|}_{\mathsf{PoS}_\circ}$ uses a label ℓ_i as (part of an) input to a query or the thread $\mathsf{A}^2_{\mathsf{PoS}_\circ}(S_0, i)$ outputs ℓ_i as its guess $\ell_{\mathsf{guess}} = \ell_i$ (note that then $i \in V'_C$) before this label ℓ_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 $\mathsf{A}^{\|}_{\mathsf{PoS}_\circ}$'s execution before they have been computed, i.e., received as output on a real query.

Claim 12. *There is an adversary A_Φ that (s', t, ϵ)-wins the pebbling game $\Phi(\mathcal{G}, V_C)$ with $s' = |F|$ initial pebbles (thus $|F| \geq s$).*

Proof of Claim. Consider an A^1_Φ which choses an initial pebbling $P_0 = F$. Then A^2_Φ in round i puts a pebble on v if $\mathsf{A}^{\|}_{\mathsf{PoS}_\circ}$ received ℓ_v as output of a real query in round i. By construction this is a valid parallel black pebbling.

We claim that this A^2_Φ puts a pebble on every node in V'_C in t steps or less, and thus (s, t, ϵ)-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 $\ell_{\mathsf{guess}} = \ell_c$ output by the thread $\mathsf{A}^2_{\mathsf{PoS}_\circ}(S_0, c)$ 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^2_Φ 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 $\|S_0\| \gtrapprox |F| \cdot w$. Let

$$[\mathsf{H}_*] \in \{0,1\}^{(2^{\iota+1}-1) \times w}$$

denote the function table of $\mathsf{H}_* : \{0,1\}^{\leq \iota} \to \{0,1\}^w$.

Claim 13. *There exists an ecoding* $(\mathsf{enc}, \mathsf{dec})$ *which correctly decodes an* $2^{-\alpha}$ *fraction of the tables*

$$\Pr_{\mathsf{H}_*}[\mathsf{dec}(\mathsf{enc}([\mathsf{H}_*])) = [\mathsf{H}_*]] \geq 2^{-\alpha}$$

and the length of the encoding is

$$\|\mathsf{enc}([\mathsf{H}_*], S_0)\| \leq \|[\mathsf{H}_*]\| + \|S_0\| - |F| \cdot (w - 2(\log N + \log q_2^{\mathsf{H}}))$$

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

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

Proof of Claim. The encoding $\mathsf{enc}/\mathsf{dec}$ will correctly decode all the $[\mathsf{H}_*]$ which are in $\mathcal{H}$. For this, $\mathsf{enc}([\mathsf{H}_*])$ first determines if $\mathsf{H}_* \in \mathcal{H}$, and if this is not the case outputs whatever (say the bit 0).

Let B denote the following computation: we first invoke $\mathsf{A}^{\|}_{\mathsf{PoS}_\circ}(S_0, V_C)$ followed by $\mathsf{L}_{\mathcal{G}}$, we'll denote with $q \leq N \cdot (q_2^{\mathsf{H}} + 1)$ the number of distinct H_* queries made during B's execution (at most q_2^{H} for each invocation of the N invocations of $\mathsf{A}^{\|}_{\mathsf{PoS}_\circ}$ and N more for $\mathsf{L}_{\mathcal{G}}$).

Let the list $\boldsymbol{c}$ contain all the outputs of H_* queries made during B. The outputs in $\boldsymbol{c}$ are stored in the order the queries were made, and if a query is repeated, the output is not stored.

Let $\overline{\boldsymbol{c}}$ denote the function table of H_* with the $|\boldsymbol{c}|$ w-bit entries that are in $\boldsymbol{c}$ removed.

Note that given $\boldsymbol{c}, \overline{\boldsymbol{c}}, S_0, V_{C'}$ we can recover $[\mathsf{H}_*]$ by running B using $\boldsymbol{c}$ to answer all the oracle queries. After this, we have learned all the inputs corresponding to the outputs stored in $\boldsymbol{c}$, and thus know which queries were deleted from $[\mathsf{H}_*]$ to get $\overline{\boldsymbol{c}}$. Thus now we can recover all of $[\mathsf{H}_*]$. We haven't compressed anything yet (as $\|\boldsymbol{c}\| + \|\overline{\boldsymbol{c}}\| = \|[\mathsf{H}_*]\|$, or as all elements in those sets and the table are w bit strings, equivalently $|\boldsymbol{c}| + |\overline{\boldsymbol{c}}| = |[\mathsf{H}_*]|$). Next we'll show how to compress $\boldsymbol{c}$ into a smaler $\boldsymbol{c}_F$ which, with some short extra information $\boldsymbol{b}_F$, 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 $\ell_i = \mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)})$ is made (the only reason we invoke $\mathsf{L}_{\mathcal{G}}$ 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 ℓ_i as part of some query input. Let $\boldsymbol{c}_F$ denote $\boldsymbol{c}$, but with the $|F|$ entries corresponding to the real queries of $i \in F$ deleted. With this $\boldsymbol{c}_F$ 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 ℓ_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 $\boldsymbol{b}_F$ denote a string encoding this information, we now define the encoding as

$$\mathsf{enc}([\mathsf{H}_*]) = (S_0, \boldsymbol{c}_F, \boldsymbol{b}_F, \overline{\boldsymbol{c}})$$

The decoding $\mathsf{dec}(S_0, \boldsymbol{c}_F, \boldsymbol{b}_F, \overline{\boldsymbol{c}})$ reconstructs $[\mathsf{H}_*]$ as outlined above. As

$$\begin{aligned}\|\boldsymbol{c}_F\| + \|\overline{\boldsymbol{c}}\| &= \|[\mathsf{H}_*]\| - w \cdot |F| \\ \|\boldsymbol{b}_F\| &\leq |F| \cdot 2\log q' \leq |F| \cdot 2(\log N + \log q)\end{aligned}$$

the encoding length is

$$\|\mathsf{enc}([\mathsf{H}_*], S_0)\| \leq \|S_0\| + \|[\mathsf{H}_*]\| - |F| \cdot (w - 2(\log N + \log q))$$

as claimed. □

■

8 PoS Security of PoCS_ϕ and PoR

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

8.1 The Labelling Games $\Lambda_{\mathsf{PoCS}_\phi}$ and Λ_{PoR}

Let $\mathsf{PoXX} \in \{\mathsf{PoCS}_\phi, \mathsf{PoR}\}$. 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 $\mathsf{H}_* : \{0,1\}^{\leq \iota} \to \{0,1\}^w, \iota = (\delta + 2) \cdot w$. The game is played by an adversary $\mathsf{A}_{\mathsf{PoXX}} = \{\mathsf{A}^1_{\mathsf{PoXX}}, \mathsf{A}^2_{\mathsf{PoXX}}\}$.

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

$$\ell_i = \begin{cases} \mathsf{H}_{*,\phi_{\boldsymbol{d}}}(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) & \text{if } i \in V \setminus V_C \\ \mathsf{H}_{*,\phi_{\boldsymbol{d}}}(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \oplus d_{\tilde{i}} & \text{if } i \in V_C \end{cases}$$

(If $\mathsf{PoXX} = \mathsf{PoCS}_\phi$) $\boldsymbol{\ell} = \{\ell'_i\}_{i \in V_C}$ where

$$\begin{aligned}\forall i \in V \ &:\ \ell_i = \mathsf{H}_{*,\phi_{\boldsymbol{d}}}(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \\ \forall i \in V_C \ &:\ \ell'_i = \ell_i \oplus d_{\tilde{i}}\end{aligned}$$

$\mathsf{A}^1_{\mathsf{PoXX}}$ 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.

$\mathsf{A}^2_{\mathsf{PoXX}}$ gets as input S_0 and the challenge c. It then proceeds in rounds, starting at round 1.

In round i, $\mathsf{A}_{\mathsf{PoXX}}$ gets as input its state S_{i-1}. It can either decide to stop the game by outputting a single guess ℓ_{guess} (for ℓ_c in PoR or ℓ'_c in PoCS_ϕ), or it can make one parallel oracle query: on query $(x_1, \ldots, x_{q_i})$ it receives $(y_1, \ldots, y_{q_i})$ where $y_i = \mathsf{H}_*(x_i)$. 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 14 (hardness of the games $\Lambda_{\mathsf{PoXX}} \in \{\Lambda_{\mathsf{PoR}}, \Lambda_{\mathsf{PoCS}_\phi}\}$)**.** *For $m, t, q_1^{\mathsf{H}}, q_2^{\mathsf{H}}, w \in \mathbb{N}$ and $p_{\mathsf{H}}, \epsilon \in [0, 1]$, we say $\mathsf{A}_{\mathsf{PoXX}} = (\mathsf{A}^1_{\mathsf{PoXX}}, \mathsf{A}^2_{\mathsf{PoXX}})$ does $(m, t, \epsilon, p_{\mathsf{H}}, q_1^{\mathsf{H}}, q_2^{\mathsf{H}})$-win the labelling game $\Lambda_{\mathsf{PoXX}}(\mathcal{G}, V_C, w)$ (as defined above) if for all but a p_{H} fraction of H_* the following holds: for at most an ϵ fraction of challenges c, $\mathsf{A}^2_{\mathsf{PoXX}}$ correctly guesses c's label (i.e., $\ell_{\mathsf{guess}} = \ell_c$) in round t or earlier. Moreover $\mathsf{A}^1_{\mathsf{PoXX}}$ and $\mathsf{A}^2_{\mathsf{PoXX}}$ make at most q_1^{H} and q_2^{H} queries to H_*, respectively. We say $\Lambda_{\mathsf{PoXX}}(\mathcal{G}, V_C, w)$ is $(m, t, \epsilon, p_{\mathsf{H}}, q_1^{\mathsf{H}}, q_2^{\mathsf{H}})$-hard if no such $\mathsf{A}_{\mathsf{PoXX}}$ exists.*

8.2 Φ Hardness Implies Λ_{PoR} and $\Lambda_{\mathsf{PoCS}_\phi}$ Hardness

Theorem 15 (hardness of Φ implies hardness of $\Lambda_{\mathsf{PoR}} \& \Lambda_{\mathsf{PoCS}_\phi}$)**.** *For any $\alpha > 0$, if the pebbling game $\Phi(\mathcal{G}, V_C)$ is (s, t, ϵ)-hard, then the labelling game $\Lambda_{\mathsf{PoR}}(\mathcal{G}, V_C, w)$ and also the labelling game $\Lambda_{\mathsf{PoCS}_\phi}(\mathcal{G}, V_C, w)$ is $(m, t, \epsilon, 2^{-\alpha} + {q_1^{\mathsf{H}}}^2/2^w, q_1^{\mathsf{H}}, q_2^{\mathsf{H}})$-hard where*

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

Before we get to proof of this theorem, let us state what security it implies for PoR and PoCS_ϕ using the hardness of Φ as stated in Lemma 8.

Corollary 16 (of Thm. 10 and Lem. 8)**.** *For $\mathcal{G}^{\epsilon'}_{4N}, V_C$ as in Lemma 8, and any $c_e \in [0, 1]$,*

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

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

Proof. We assume the reader is familiar with the proof of Theorem 10, as we will only explain how that proof needs to be adapted.

The proof of Theorem 10 goes through almost unchanged for $\mathsf{PoXX} \in \{\mathsf{PoR}, \mathsf{PoCS}_\phi\}$ instead of $\mathsf{PoS}_\circ$, the point where it fails is when we need to compress fresh labels. In the proof of Theorem 10 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 PoR. Now even if the encoding enc observes a fresh label ℓ_i when invoking $\mathsf{A}^{\|}_{\mathsf{PoR}}$ (which is defined analogous to $\mathsf{A}^{\|}_{\mathsf{PoS}_\circ}$ in the proof of Theorem 10), it's not clear how to compress one entry of $\mathsf{H}_{*,\phi_{\boldsymbol{d}}}$'s function table as now

$$\ell_i = \mathsf{H}_{*,\phi_{\boldsymbol{d}}}(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \oplus d_{\tilde{i}}$$

only provides an output that is blinded with $d_{\tilde{i}}$. If we could make sure the encoding and decoding $\mathsf{enc}/\mathsf{dec}$ knew the $\boldsymbol{d}$, this problem would disappear.

We fix this problem as follows. We define $\mathsf{A}^{\|}_{\mathsf{PoR}}$ analogous to $\mathsf{A}^{\|}_{\mathsf{PoS}_\circ}$, i.e., it runs $\mathsf{A}^2_{\mathsf{PoR}}(S_0, c)$ on all challenges $c \in V_C$ in parallel. But additionally, at the very beginning (before invoking the $\mathsf{A}^2_{\mathsf{PoR}}$'s), it invokes $\mathsf{A}^1_{\mathsf{PoR}}$, but only runs it to the point where the commitment $\phi_{\boldsymbol{d}}$ is received as on output of H_* (recall we assume $\mathsf{A}^1_{\mathsf{PoR}}$ follows the protocol, so this commitment must be computed at some point).

This way enc/dec, wo invoke $\mathsf{A}^{\|}_{\mathsf{PoR}}$, learn the entire $\boldsymbol{d}$, as it can be extracted from the H_* queries leading to $\phi_{\boldsymbol{d}}$. At the same time $\mathsf{A}^1_{\mathsf{PoR}}$ will almost certainly not have made any $\mathsf{H}_{*,\phi_{\boldsymbol{d}}}(\cdot) = \mathsf{H}_*(\phi_{\boldsymbol{d}}, \cdot)$ queries as $\phi_{\boldsymbol{d}}$ is uniform and we stop executing $\mathsf{A}^1_{\mathsf{PoR}}$ once $\phi_{\boldsymbol{d}}$ is received. This is necessary, so a label that was fresh without running $\mathsf{A}^1_{\mathsf{PoR}}$ first, will still be fresh if we do run $\mathsf{A}^1_{\mathsf{PoR}}$.

The above argument works as long as $\mathsf{A}^1_{\mathsf{PoR}}$ doesn't find a collision in H_* (otherwise we can't extract a unique $\boldsymbol{d}$). For this reason in the theorem security holds only for a slightly smaller $p_{\mathsf{H}} = 2^{-\alpha} + {q_1^{\mathsf{H}}}^2/2^w$ fraction of the H_* than the $p_{\mathsf{H}} = 2^{-\alpha}$ fraction we got for the $\Lambda_{\mathsf{PoS}_\circ}$ game in Theorem 10. ■

References

[AAC+17] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 357–379. Springer, 2017.

[ABH17] 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.

[ABP17] 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 II*, volume 10211 of *LNCS*, pages 3–32. Springer, Heidelberg, May 2017.

[ABP18] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In *EUROCRYPT*, Lecture Notes in Computer Science, 2018.

[ACP+17] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 33–62. Springer, Heidelberg, May 2017.

[BCK+14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *STOC*, pages 857–866. ACM, 2014.

[BKLS16] 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.

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of useful work. *IACR Cryptology ePrint Archive*, 2017:203, 2017.

[bur] Burstcoin. `http://burstcoin.info`.

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

[CKWN16] 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.

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

[DGK17] 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, May 2017.

[DNW05] 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.

[DPS16] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. `http://eprint.iacr.org/2016/919`.

[DTT10] 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.

[EGS75] Paul Erdoes, Ronald L. Graham, and Endre Szemeredi. On sparse graphs with dense long paths. Technical report, Stanford, CA, USA, 1975.

[KN] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake.

[KPS13] 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.

[KRDO17] 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.

[Lab17] Protocol Labs. Filecoin: A decentralized storage network. https://filecoin.io/filecoin.pdf, 2017.

[Mic16] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[MJS+14] 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.

[MO16] Tal Moran and Ilan Orlov. Rational proofs of space-time. Cryptology ePrint Archive, Report 2016/035, 2016. `https://eprint.iacr.org/2016/035`.

[MZ17] 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.

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

[PTC77] 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.

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

[The14] 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 critisized 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.[7]

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 general idea is to choose the winner proportional to the fraction of coins they hold. At first, this idea looks extremely promising, but it seems to be really difficult to actually realize it in a secure and efficient way. Some early ad-hoc implementations of PoStake based blockchain include Peercoin [KN] and NXT [The14]. More recent proposals come with security proofs in various models [Mic16, DPS16, KRDO17], but those protocols are fairly complicated, and basically run a full blown 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.

Space as a Resource. After time, space is the best investigated resource in computational complexity, thus it's a natural idea to somehow try to use disk space as a resource for mining, which 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 [MJS+14] is a proposal which 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 [bur], this design is poorly document, but it seems to have security and efficiency issues.[8] Another suggestion are "proofs of space"

[7] 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 [CKWN16].

[8] [PPK+15, 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.

(PoS), which are the topic of this paper and we'll discuss them in more detail below in §A.2.

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 [BRSV17] 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. This holds the potential for blockchain designs which are even more sustainable than pure PoS-based blockchains.

A.2 Proofs of Space (PoS)

Proofs of space (PoS) [DFKP15] 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 [PPK+15], a recent ongoing effort which combines PoS with some type of proofs of sequential work is the chia network [chi17].

A PoS [DFKP15] 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 initalized 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.[9]

To date two very different types of PoS have been suggested. PoS-based on hard to pebble graphs [DFKP15, RD16] and PoS-based on inverting random functions [AAC+17], 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 the [AAC+17] PoS to PoR and PoCS, this would be very interesting as although the [AAC+17] 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

`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 [PPK+15]). 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.

[9] Though it might be meaningful to intentionally make the initialization phase more expensive. Some ideas how to do that without at the same time making verification slower are discussed in [MO16].

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 entire proof must somehow be made non-interactive.[10]

B $\mathsf{PoCS_F}$, a PoCS with Efficient Updates

In §B.1 we first give the specification of $\mathsf{PoCS_F}$, our proof of catalytic space that allows for efficient updates of the catalytic data. The security game $\Lambda_{\mathsf{PoCS_F}}$ capturing the PoS security of $\mathsf{PoCS_F}$ is then defined in §B.2. In §B.3 we prove Theorem 18, which states that hardness of the pebbling game Φ implies hardness of $\Lambda_{\mathsf{PoCS_F}}$. The PoS security of $\mathsf{PoCS_F}$ is then derived as Corollary 19 in §B.3.

B.1 The Protocol $\mathsf{PoCS_F}$

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

κ, λ : The $\kappa \in \mathbb{N}$ specifies the rate (i.e., $\|\boldsymbol{d}\|/\|\boldsymbol{\ell}\|$ is $\kappa/(\kappa+1)$), $\lambda \stackrel{\mathsf{def}}{=} (\kappa+1)\cdot w$ is the label length.

$\mathcal{G}$: A DAG $\mathcal{G} = (V, E)$ with a designated set $V_C \subseteq V, |V_C| = N$ of challenge nodes.

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

F : A family of random invertible functions $\{\mathsf{F}_\chi\}_{\chi \in \{0,1\}^w}$.

$$\mathsf{F}_\chi : \{0,1\}^{\lambda - w} \to \{0,1\}^{\lambda} \ , \ \mathsf{F}_\chi^{-1} : \{0,1\}^{\lambda} \to \{0,1\}^{\lambda - w} \cup \perp$$

Initialization. V picks a random statement χ and sends it to P. P chooses any data $\boldsymbol{d} = \{d_i\}_{i \in [N]}, d_i \in \{0,1\}^{\kappa \cdot w}$ and then computes the file to store as

$$\boldsymbol{\ell} := \mathsf{E}_{\mathsf{PoCS_F}}(\chi, \boldsymbol{d}) \quad \text{(cf. §4.4)}$$

P computes $(\phi_{\boldsymbol{\ell}}, \phi_{\boldsymbol{\ell}}^+) := \mathsf{commit}^{\mathsf{H}_\chi}(\boldsymbol{d})$, sends $\phi_{\boldsymbol{\ell}}$ to V_C and locally stores $\phi_{\boldsymbol{\ell}}^+$. This concludes the initialization if we assume P is honest (cf. Remark 3 in §4.5). Thus V stores $\chi, \phi_{\boldsymbol{\ell}}$, the prover stores $\chi, \phi_{\boldsymbol{\ell}}^+$ and $\boldsymbol{\ell} = \{\ell_i\}_{i \in V_C}, \ell_i \in \{0,1\}^{\lambda}$.

[10] Concretely, for subtle security reasons Spacemint [PPK+15] (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 [AAC+17] will not require any such commitments.

Proof execution. The protocol where $\mathsf{P}(\boldsymbol{\ell}, \chi, \phi^{+}_{\boldsymbol{\ell}})$ convinces $\mathsf{V}(\chi, \phi_{\boldsymbol{\ell}})$ that it stores $\boldsymbol{\ell}$ is very simple. V samples a few nodes from the challenge set $\boldsymbol{c} = (c_1, \ldots, c_\mu) \subset [N]$ at random, and sends the challenge $\boldsymbol{c}$ to P. P sends openings $\boldsymbol{o} := \mathsf{open}(\boldsymbol{\ell}, \phi^{+}_{\boldsymbol{\ell}}, \boldsymbol{c})$ to the labels $\{\ell_{\tilde{i}}\}_{i \in \boldsymbol{c}}$ to V, who then accepts iff $\mathsf{verify}(\phi_{\boldsymbol{\ell}}, \boldsymbol{c}, \boldsymbol{o}) = 1$.

B.2 The Labelling Game $\Lambda_{\mathsf{PoCS_F}}(\mathcal{G}, V_C, w, \kappa)$

The game is parameterized by a DAG $\mathcal{G} = (V, E)$, a subset $V_C \subseteq V$ of $|V_C| = N$ challenge nodes, a block size w and a parameter κ. A function (with range $\lambda = (\kappa + 1) \cdot w$)

$$\mathsf{H}_* : \{0,1\}^{\leq \iota} \to \{0,1\}^{\lambda}$$

and an invertible function

$$\mathsf{F}_* : \{0,1\}^{\lambda - w} \to \{0,1\}^{\lambda} \ , \ \mathsf{F}_*^{-1} : \{0,1\}^{\lambda} \to \{0,1\}^{\lambda - w} \cup \bot$$

It is played by an adversary given as a pair $\mathsf{A}_{\mathsf{PoCS_F}} = \{\mathsf{A}^1_{\mathsf{PoCS_F}}, \mathsf{A}^2_{\mathsf{PoCS_F}}\}$.

initialization: Player $\mathsf{A}^1_{\mathsf{PoCS_F}}$ is given oracle access to H_* and $\mathsf{F}_*/\mathsf{F}_*^{-1}$. It can choose any data $\boldsymbol{d} = \{d_i\}_{i \in V_C}$, $d_i \in \{0,1\}^{\lambda - w}$. This then defines labels $\boldsymbol{\ell}$ to store as in eq.(6) (but using $\mathsf{H}_*, \mathsf{F}_*$ instead of $\mathsf{H}_\chi, \mathsf{F}_\chi$), concretely $\boldsymbol{\ell} = \{\ell'_i\}_{i \in V_C}$ where

$$\ell_i = \begin{cases} \mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) & \text{if } i \in V_C \\ \mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)})_{|w} & \text{if } i \in V \setminus V_C \end{cases} \tag{9}$$

and

$$\ell'_i = \ell_i \oplus \mathsf{F}_*(d_{\tilde{i}}) \text{ for } i \in V_C$$

$\mathsf{A}^1_{\mathsf{PoCS_F}}$ 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.

$\mathsf{A}^2_{\mathsf{PoCS_F}}$ gets as input S_0 and the challenge c. It then proceeds in rounds, starting at round 1.

In round i, $\mathsf{A}^2_{\mathsf{PoCS_F}}$ gets as input its state S_{i-1}. It can either decide to stop the game by outputting a single guess ℓ_{guess} (for ℓ_c), or it can make one parallel oracle query to $\mathsf{H}_*, \mathsf{F}_*, \mathsf{F}_*^{-1}$: on inputs three tuples of queries $\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y}$, it gets $\mathsf{H}_*(\boldsymbol{x}), \mathsf{F}_*(\boldsymbol{x}'), \mathsf{F}_*^{-1}(\boldsymbol{y})$. 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 17 (hardness of the game $\Lambda_{\mathsf{PoCS_F}}$)**.** *For $m, t, q_2^{\mathsf{H}}, q_1^{\mathsf{F}}, q_2^{\mathsf{F}}, w, \in \mathbb{N}$ and $p_{\mathsf{H,F}}, \epsilon \in [0,1]$, we say $\mathsf{A}_{\mathsf{PoCS_F}} = (\mathsf{A}^1_{\mathsf{PoCS_F}}, \mathsf{A}^2_{\mathsf{PoCS_F}})$ does $(m, t, \epsilon, p_{\mathsf{H,F}}, q_2^{\mathsf{H}}, q_1^{\mathsf{F}}, q_2^{\mathsf{F}})$-win the labelling game $\Lambda_{\mathsf{PoCS_F}}(\mathcal{G}, V_C, w, \kappa)$ (as defined above) if for all but a $p_{\mathsf{H,F}}$ fraction of $\mathsf{H}_*, \mathsf{F}_*$ tuples the following holds: for at most an ϵ fraction of challenges c, $\mathsf{A}^2_{\mathsf{PoCS_F}}$ correctly guesses c's label (i.e., $\ell_{\mathsf{guess}} = \ell_c$) in round t or earlier. Moreover $\mathsf{A}^1_{\mathsf{PoCS_F}}$ makes at mos q_1^{F} queries to $\mathsf{F}_*/\mathsf{F}_*^{-1}$ and and $\mathsf{A}^2_{\mathsf{PoCS_F}}$ make at most q_2^{H} queries to H_* and q_2^{F} queries to $\mathsf{F}_*/\mathsf{F}_*^{-1}$. We say $\Lambda_{\mathsf{PoCS_F}}(\mathcal{G}, V_C, w, \kappa)$ is $(m, t, \epsilon, p_{\mathsf{H,F}}, q_2^{\mathsf{H}}, q_1^{\mathsf{F}}, q_2^{\mathsf{F}})$-hard if no such $\mathsf{A}_{\mathsf{PoCS_F}}$ exists.*

B.3 Φ Hardness Implies $\Lambda_{\mathsf{PoCS_F}}$ Hardness

Theorem 18 (hardness of Φ implies hardness of $\Lambda_{\mathsf{PoCS_F}}$)**.** *For any $\alpha > 0$, if the pebbling game $\Phi(\mathcal{G}, V_C)$ is (s,t,ϵ)-hard, then the labelling game $\Lambda_{\mathsf{PoCS_F}}(\mathcal{G}, V_C, w, \kappa)$ is $(m, t, \epsilon, 2^{-\alpha} + \frac{q^2}{2^{w/2}}, q_2^{\mathsf{H}}, q_1^{\mathsf{F}}, q_2^{\mathsf{F}})$-hard where*

$$q \stackrel{\mathsf{def}}{=} 2(q_1^{\mathsf{F}} + \log N \cdot q_2^{\mathsf{F}}) \quad , \quad \lambda \stackrel{\mathsf{def}}{=} (\kappa+1) \cdot w$$

$$m = s \cdot (\lambda - w/2 - 2 \cdot \log(N \cdot q_2^{\mathsf{H}})) - \alpha \tag{10}$$

Before we get to proof of this theorem, let us state what security it implies for $\mathsf{PoS}_\circ$ using the hardness of Φ as stated in Lemma 8.

Corollary 19 (of Thm. 10 and Lem. 8)**.** *For $\mathcal{G}_{4N}^{\epsilon'}, V_C$ as in Lemma 8, any $c_e \in [0,1]$ and q as in the theorem above,*

$$\Lambda_{\mathsf{PoCS_F}}(\mathcal{G}_{4N}^{\epsilon'}, V_C, w, \kappa) \quad \textit{is} \quad (m, t, \epsilon, 2^{-\alpha} + \frac{q^2}{2^{w/2}}, q_2^{\mathsf{H}}, q_1^{\mathsf{F}}, q_2^{\mathsf{F}})\textit{-hard with}$$

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

Proof. The proof of this theorem follows the same lines as the proof of Theorem 10. We assume the reader is familiar with the proof of Theorem 10 and will only discuss how to adapt the proof. In the proof of Theorem 10, for every fresh $i \in F$, the encoding extracted a label ℓ_i before it was received as output of the oracle query $\ell_i = \mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)})$, and thus was able to compress H_* by (almost) $\|\ell_i\| = w$ bits. Here we can do the same for any $i \in F \cap (V \setminus V_C)$, but for $i \in F \cap V_C$ the label is defined as

$$\ell_i = \mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \oplus \mathsf{F}_*(d_{\tilde{i}}) \ .$$

Thus when the encoding extracts a fresh label $\ell_i \in \{0,1\}^\lambda$ for $i \in F \cap V_C$, it only learns the oracle output $\mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)})$ blinded with $\mathsf{F}_*(d_{\tilde{i}})$, and the encoding can't directly compress this value.

In the proof of Theorem 15 we encountered a similar problem. There we solved this problem by having the encoding initially invoke $\mathsf{A}^1_{\mathsf{PoXX}}$, from which we could then extract all of $\boldsymbol{d}$. It was crucial to stop the invocation of $\mathsf{A}^1_{\mathsf{PoXX}}$ after it queried for $\phi_{\boldsymbol{d}}$, because at this point all of $\boldsymbol{d}$ could be extracted, while almost certainly no queries computing labels were made by $\mathsf{A}^1_{\mathsf{PoXX}}$ as those are computed using $\mathsf{H}_{*,\phi_{\boldsymbol{d}}}$, which thus requires knowledge of $\phi_{\boldsymbol{d}}$. Unfortunately, here we can't pull of that trick again, as now $\mathsf{A}^1_{\mathsf{PoCS_F}}$ can make all the label queries as in eq.(9) before it makes any of the F_* queries determining $\boldsymbol{d}$, thus we'll later not be able to compress the labels.

We could additionally provide $d_{\tilde{i}} \in \{0,1\}^{\lambda - w}$ to the encoding, but then we'll only compress the $\lambda = (\kappa+1) \cdot w$ bit long label by at most $|\ell_i| - |d_{\tilde{i}}| = w$ bits, which isn't interesting.[11]

[11] I.e., the conclusion we get "compressing" like this is the same as if we use the normal labelling game, and let the prover store the data in the clear.

The key idea to overcome this problem is to first replace the random invertible function $\mathsf{F}_* : \{0,1\}^{\lambda-w} \to \{0,1\}^\lambda$ with a composed function. For this we pick random functions

$$\mathsf{f} : \{0,1\}^{\lambda-w} \to \{0,1\}^{w/2} \quad , \quad \mathsf{g} : \{0,1\}^{w/2} \to \{0,1\}^\lambda$$

and define

$$\tilde{\mathsf{F}}(x) = \mathsf{g}(\mathsf{f}(x)) \ . \tag{11}$$

Let q denote an upper bound on the number of F_* (or now $\tilde{\mathsf{F}}$) queries made during encoding, recall that the number is determined by

$$q = (q_1^{\mathsf{F}} + N \cdot q_2^{\mathsf{F}})$$

We'll observe a collision on f with probability at most $q^2/2^{w/2}$, and conditioned on no such collision happening, the new game – where we replaced F_* with $\tilde{\mathsf{F}}$ – behaves like the original $\Lambda_{\mathsf{PoCS_F}}$ game.

If such a collision happens, the encoding declares failure, that's why in the statement of the Theorem, the fraction of $\mathsf{H}_*, \mathsf{F}_*$ tuples for with the games fail to be hard is $2^{-\alpha} + \frac{q^2}{2^{w/2}}$ (not just $2^{-\alpha}$ as we had for $\mathsf{E}_{\mathsf{PoS}_\circ}$).

If we now observe a label

$$\ell_i = \mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \oplus \tilde{\mathsf{F}}(d_{\tilde{i}}) = \mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) \oplus \mathsf{g}(\mathsf{f}(d_{\tilde{i}}))$$

we can recover the random oracle output $\mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)})$ given just the short $\mathsf{f}(d_i) \in \{0,1\}^{w/2}$ (as $\mathsf{H}_*(i, \boldsymbol{\ell}_{\mathsf{parents}(i)}) = \ell_i \oplus \mathsf{g}(\mathsf{f}(d_{\tilde{i}}))$), this $w/2$ bit extra hint is the reason for the $-w/2$ term in eq.(10).

There is one more difficulty we have to solve. As as the adversary gets access not just to F_*, but also F_*^{-1}, we have to specify how to answer $\tilde{\mathsf{F}}^{-1}$ queries. On a query $\tilde{\mathsf{F}}^{-1}(y)$ for some y that has not been observed as output of $\tilde{\mathsf{F}}$ before, we simply output $\perp$, this almost perfectly simulates F_*^{-1} as the output domain of F_* is sparse (only a tiny 2^{-w} fraction of values in the range $\{0,1\}^\lambda$ have a pre-image).

On a query $\tilde{\mathsf{F}}^{-1}(y)$ where the query $\tilde{\mathsf{F}}(x) = y$ has been observed before, we give the corresponding output x. This is a problem if the $\tilde{\mathsf{F}}(x) = y$ query was made during the first phase by $\mathsf{A}^1_{\mathsf{PoCS_F}}$, but the inverse query $\tilde{\mathsf{F}}^{-1}(y)$ is made by an instantiation of $\mathsf{A}^2_{\mathsf{PoCS_F}}$ during the encoding, as now we'll have to store the corresponding x value as part of the encoding. But note that if this happens, we have observed a $y \in \{0,1\}^\lambda$ from the function table of g, and thus can compress it, which "pays" for the $x \in \{0,1\}^{\lambda-w}$ bits we need to store. ■