Separating the Pebbling Model from the Random Oracle Model

Susanna F. de Rezende
Lund University
susanna.rezende@cs.lth.se

David Engström
Lund Univeristy
david.engstrom@cs.lth.se

Leonid Reyzin
Boston University
reyzin@bu.edu

May 21, 2026

Abstract

The security of cryptographic constructions that enforce resource usage, such as Proofs of Work or Proofs of Space, is often shown in the random oracle model. This model restricts the class of possible adversaries, because it assumes that the adversary can access some function RO only as a black box, via queries. When the resource in question is space, the random oracle model is often further idealized by assuming that the outputs of RO are usable only in a black box manner: they are either stored whole or discarded whole by the adversary, and are never computed upon, except when provided as inputs to RO. In this idealization, they are often called "pebbles," and space usage is counted in terms of pebbles stored.

In some cases, it is known that the pebbling model does not add further restrictions on the adversary, because the bit strings that correspond to the pebbles can actually be extracted from the adversary's memory. In other cases, this question has been open for over a decade.

We resolve the open question by showing that the pebbling model does not realistically model adversarial capabilities in two important cases. Specifically, we construct a family of Proofs of Space and a family of Memory-Hard Functions in the pebbling model for which an algorithm that is allowed to treat outputs of RO as bit strings and compute upon them (simply by XORing subsets of them) can be significantly more efficient that an algorithm limited to pebbling.

1 Introduction

1.1 The Random Oracle Model with Pebbling

The research area of moderately hard computation (such as proofs of space, proofs of work, memory-hard functions, etc.) attempts to find computational problems with roughly matching upper and lower bounds on computational resources required. Given the difficulty of proving lower bounds on computational resources for arbitrary, potentially adversarial, algorithms, most solutions rely on random oracle model. As is common in other research areas, time complexity is measured in the number of oracle queries, and each query computation—even if performed by adversary—is modeled as atomic.

However, moderately hard computation is often concerned not only with time complexity, but also with space complexity, and with the tradeoff between the two. Proofs of lower bounds on space complexity often model not only random oracle *computations*, but also random oracle *outputs*, as atomic. In this model, only two choices are available to an adversary for any random oracle output: store it whole or discard it. This model is often called the *pebbling* model, where a pebble refers to a random oracle output that is stored. These pebbles are placed on a DAG, in which nodes correspond to random oracle queries, and a directed edge (u, v) shows the flow of information: namely, that that an output of query u becomes an input to query v. One is thus allowed to place a pebble on a given node only if all of its immediate predecessors are pebbled.

The pebbling model is even more idealized than the random oracle model, as a real adversary can compute arbitrarily on the bits of the random oracle outputs. The heuristic justification for why it makes sense to model random oracle outputs as atomic pebbles is the following: these outputs are uniformly random strings that are either used as inputs to subsequent random oracle queries or sent to a verifier. In either case, they cannot be compressed (because they are random) and are useful only if they are unmodified.

1.2 Is the Pebbling Model Secure?

The question of whether the pebbling model is equivalent to the random oracle model has been addressed in different settings. In 2005, Dwork, Naor, and Wee [DNW05] showed this equivalence in the context of graph-based memory bound functions, by demonstrating that a graph pebbling can be ex post facto reconstructed from the random oracle queries made by the adversary. Their techniques were used by Dziembowski, Kazana, and Wichs and by Karvelas and Kiayias in the context of security against memory-limited adversaries [DKW11b; DKW11a; KK14] and by Ateniese, Bonacina, Faonio, and Galesi [ABFG14] for proofs of transient space. Thus, in these cases the restrictive pebbling model of the adversary was not necessary for the final result; it was merely a stepping stone.

However, establishing this equivalence in other contexts has proved elusive. In 2009, Percival proposed the first memory-hard function (MHF) called "scrypt" [Per09]. When Alwen and Serbinenko formalized the security of MHFs [AS15], they were able to prove that the pebbling model is equivalent to the random oracle model only for a restricted category of MHF constructions (so-called data-independent MHFs, or iMHFs, in which the computation graph did not depend on the input); this category does not include scrypt itself. The following year, Alwen, Chen, Kamath, Kolmogorov, Pietrzak, and Tessaro [ACKKPT16b] were able to prove the security of scrypt in a slightly relaxed pebbling model, which allowed the adversary to "entagle" pebbles by storing XORs

of random oracle outputs (for emphasis, we will call a computation that does not entagle pebbles a "pure" pebbling). Intriguingly and counterintuitively, they showed an example game in which an entangled pebbler could save a factor of 1.5 in cumulative memory complexity versus the pure pebbler, even though the games themselves never asked for linear combinations of random oracle outputs [ACKKPT16a, Appendix A]. They conjectured that no example can improve this factor of 1.5 to superconstant. The equivalence of the pebbling model and the random oracle model specifically for scrypt was finally shown by Alwen, Chen, Pietrzak, Reyzin, and Tessaro [ACPRT17], but a more general result for dMHFs is still unknown.

Similarly, in their paper that defined and constructed the first proofs of space[1], Dziembowski, Faust, Kolmogorov, and Pietrzak were able to prove their result only in the pebbling model, and conjectured that it is equivalent to the random oracle model [DFKP15, Section 3.1]. Pietrzak [Pie19, Section 7.2] was able to prove this conjecture for an alternative definition of proof of space, namely the one that counts parallel computation time (i.e., number of steps where unlimited parallel random oracle queries are allowed in each step) rather than sequential computation time (i.e., total number of random oracle queries, which corresponds roughly to computational cost).

1.3 Our Results

In this paper, we refute the conjectures of [DFKP15] and [ACKKPT16b] by demonstrating a proof of space (PoS) and a data-dependent memory hard function (dMHF) that are significantly more secure against a pure pebbling adversary than against the more general random-oracle model adversary. Our more general adversary will compute on random-oracle outputs in only a limited way, simply by XORing subsets of them. We will therefore call this model "the entangled pebbling model," following [ACKKPT16b].

PoS Result Before stating our separation for proofs of space, we recall what they are (a more detailed exposition is in Section 2.1). Initialization consists of labeling a large DAG T using a random oracle RO at each node, applied to the labels of the node's predecessors. After initialization, the prover is supposed to store labels for some large subset Y of the nodes of T. The verifier will periodically ask for labels of random nodes in Y. To build a good PoS, one needs to design T so that a prover who is storing only some fraction r_{space} of Y will need to do redo a large fraction r_{hardness} of the initial computation to answer such a query.

Our separation for PoS can be stated as follows (see Theorem 4.1 for the precise statement).

Theorem 1.1 (Informal)**.** *For any k and d and any r_{space} and $r_{\text{hardness}} < 1 - r_{\text{space}}$, there is a PoS such that*

- *any adversary limited to pure pebbling and storing r_{space} fraction of the required storage will have to redo r_{hardness} fraction of the initial computation to answer the verifier;*
- *there is an adversary using entangled pebbles who stores only $\frac{1}{d}$ fraction of the require storage who has to redo only a $\frac{1}{k}$ fraction of the initial computation to answer the verifier.*

We note that while $r_{\text{hardness}} < 1 - r_{\text{space}}$ is a restriction on achievable parameters, existing constructions generally do not do much better. For a concrete example, a proof of space with

[1]More precisely, we are referring to proofs of *persistent* space, as opposed to proof of *transient* space mentioned above [ABFG14].

conjectured $r_{\text{space}} = 0.8$ and $r_{\text{hardness}} \approx 0.018$ from [Fis19; GN23] has seen significant practical deployment in the Filecoin blockchain [Pro23]. However, Reyzin [Rey24] showed a construction in which it is possible to get both r_{hardness} and r_{space} close to 1 if one is willing to grow the graph size (and also showed $r_{\text{hardness}} \approx 0.2$ in the deployed construction).

While our separation is stated generally, we also provide an example of a PoS with concrete parameters not too different from those deployed in practice, in which entangling pebbles can save a factor of 15 in work for the same space, or a factor of 5 in work and a factor of 3 in space (Section 4.4).

dMHF Result We now give a brief reminder of dMHFs in order to state our dMHF result (see Section 2.2 for details). Let F be a function that makes queries to RO; the complexity of computing F, called cumulative memory complexity (CMC), is measured by the sum of the amount of memory used in each time step. We focus on dMHFs that can be represented by a pebbling game, as follows: the first part of the computation consists of labeling a fixed DAG G using RO, like in a PoS. The second part consists of the following step repeated t times: $\ell_{i+1} = RO(\ell_i, \ell_{y_i})$, where y_i is an integer computed from ℓ_i and ℓ_{y_i} is the label of node number y_i in G. The last computed label ℓ_t is the output. Because y_i is unknown until ℓ_i is derived, and ℓ_i is random as the output of RO, this random oracle game is modeled by the following pebbling game: the pebbler is repeatedly asked to place a pebble on a randomly chosen node y_i in G. Our dMHF result is as follows (see Theorem 5.1 for the formal statement).

Theorem 1.2 (Informal)**.** *There exists a family of pebbling-based dMHFs $\{F_n\}_{n\in\mathbb{N}}$ computable in sequential time $O(n)$ such that*

- *any computation of F_n limited to pure pebbling has cumulative memory complexity $\Omega\left(\frac{n^2}{\log n}\right)$;*
- *it is possible to compute F_n using entangled pebbling with cumulative memory complexity $O\left(\frac{n^2 \log\log n}{\log^2 n}\right)$.*

Note that in the pure pebbling model these dMHFs are close to optimal, as a time-$O(n)$ computation uses $O(n)$ memory and thus has $O(n^2)$ CMC.

1.4 Our Approach

Our main idea, described in Section 3, is to start with a base PoS $\hat{T}$ and replicate it multiple times in parallel. The gap between the pure and entangled pebbler comes from a very specific query pattern (enforced by adding some connecting nodes and edges). A single query hits one node in each of the replicas. While a random query hits a uniform node in each replica, the inter-replica pattern creates sets of nodes of size d that are worth entangling. At first glance, it may seem that entangling a set of size d cannot help save more than $1/d$ in computational complexity, because $d-1$ pure pebbles need to be computed in order to disentangle and obtain the last one. The reason our construction works is that the entangled pebbler computes many sets of $d-1$ pure pebbles at once by computing $d-1$ base PoSes; these sets are then used to help all the remaining base PoSes because they can disentangle many entangled pebbles.

This idea is used on both our PoS and our dMHF construction. The difference is in the properties of $\hat{T}$ when insufficient space is dedicated to storing it and a query arrives: in the PoS case it needs to have high cost with high probability, and in the dMHF case it needs to have high expected CMC.

2 Background and Definitions

In this paper, all computing parties are assumed to have access to a public random oracle $RO : \{0,1\}^* \to \{0,1\}^w$, where w is chosen big enough that random oracle outputs are overwhelmingly unguessable without querying the random oracle and are overwhelmingly unlikely to collide.

Unless otherwise stated we use $\log n$ to denote $\log_2 n$.

2.1 Background: Proofs of Space

To initialize a proof of space, a prover $P(x)$ starts with a short value x as input, produces a large output Y using some computation that takes time t_{init}, and sends to the verifier $V(x)$ a commitment to Y and a proof that Y was (almost) correctly computed. After that, the goal is for V to be convinced that P is using $|y|$ storage continuously. To do so, V periodically queries random portions of Y, which P returns together with the proof of their correctness with respect to the initial commitment. To be confident that P is really using the storage, we need the following property: if P is storing less than $r_{\text{space}} \cdot |Y|$, then the amount of computation required to satisfy V should be at least $r_{\text{hardness}} \cdot t_{\text{init}}$. The parameters r_{space} and r_{hardness} are called the *space ratio* and the *hardness ratio*, respectively.

DAG-based PoS Most PoS constructions are based on random-oracle DAG labeling. Let T be a single-source DAG of t_{init} nodes. x is the label of the source node; the initial computation of P consists of labeling each node i by applying RO to the labels of its immediate predecessors (i.e., sources of its incoming edges) and its index i; and Y consists of labels of s topologically last nodes of the DAG. To initialize such a construction, P computes all the node labels, proves to V that most of them are correctly computed (by vector-committing to the labels and then showing, in response to random probes by V, labels of random nodes and their immediate predecessors), and stores the labels of Y. The verifier V periodically queries the labels of nodes of Y to ensure the prover P is still storing them.

Pure Pebbling Model In the pebbling model, the adversary (whom we will call a "pure pebbler") can store some of the node labels (only whole — no partial labels, XORs of labels, etc.), which is modeled as placing some number of pebbles on nodes. If adversarial storage is less than $r_{\text{space}}|Y|$, then a query asking for some the labels of some subset of nodes $Q \subseteq Y$ may hit some unpebbled nodes, and the cost of answering such a query will be proportional to the number of nodes in T that have unpebbled paths to Q, because the labels of all these nodes need to computed in order to compute the labels of Q. A proof of space is characterized by the "soundness error" σ that measures the chances of this cost being less than $r_{\text{hardness}} t_{\text{init}}$.

For proofs of space in this paper we assume at most one random oracle call at a time, and thus measure time complexity of computation as the total number of random oracle queries (which in real life corresponds roughly to sequential time and monetary cost, and in the pebbling model corresponds to the number of pebbles placed). Our results cannot hold in the parallel computation model, as we know that for parallel computation pebbling and random oracles models are equivalent [Pie19, Section 7.2].

Definition 2.1 (Proof of Space)**.** Let T be a single-source DAG with t_{init} nodes, of which a subset Y of s nodes are designated as output nodes. Let $\mathcal{Q}$ be an efficiently samplable set, each of whose

elements Q is a subset of Y of size q. (Unless otherwise specified, $\mathcal{Q}$ consists of a singleton set for every element of Y.) We will say that $(T, \mathcal{Q})$ (or just T, when $\mathcal{Q}$ is clear from the context) is a proof of space in the pure pebbling model with space ratio r_{space}, hardness ratio r_{hardness}, and q-query soundness error σ if the following holds. Suppose r_{space} fraction of the nodes of T has pebbles placed on it. Then for a random $Q \in \mathcal{Q}$, with probability at least $1 - \sigma$, the fraction of the nodes of T that has unpebbled directed paths to Q is at least r_{hardness} (where a path is unpebbled if neither the endpoints nor the intermdiate nodes have pebbles on them). *Expected* hardness ratio ρ_{hardness} is the average of this fraction, over the choice of $Q \in \mathcal{Q}$.

Naturally, σ can be reduced to σ^ℓ by sending a batch of ℓ random choices Q from $\mathcal{Q}$. The ρ_{hardness} measure is relevant for a rational prover who is trying to decide whether to dedicate storage to Y or erase it and risk having to perform a costly computation when a query arrives.

As an ingredient in our construction, we will use the construction of Reyzin [Rey24]) that can get r_{hardness} and r_{space} arbitrarily close to 1 (by growing T) for so-called maximal hardness. Note that σ cannot be lower than r_{space} if $q = 1$ (because the query can just hit a pebble); the construction of [Rey24] has $\sigma = (1 + r_{\text{space}})/2$ for $q = 1$.

2.2 Background: Memory Hard Functions

Consider a computation that proceeds in steps. Its cumulative memory complexity (CMC) is the sum of the amount of memory used at each step. We will consider computations in the parallel random oracle model (pROM), where a single step consists of issuing some number of queries to RO and receiving all the responses at once. If $y = F(x)$ is a function that makes queries to RO, then its CMC is the lowest CMC over all possible algorithms that compute this function correctly (technically, lowest expected CMC, where expectation is over the choice of RO). A function with high CMC is called a *memory-hard function* (MHF).

Pebbling Complexity and iMHFs Suppose F is represented as a single-source single-sink n-node DAG G, where x is the label of the source, y is the label of the sink, and the label of each node i is computed by applying RO to the labels of its immediate predecessors and its index i. We say that such F is *data-independent*. In that case, a natural computation of F in the pROM, which simply computes labels in order by querying the random oracle, and either stores or discards whole labels, can then be represented as a *parallel pebbling game*, where a player starts with a pebble on the source, and at each step is allowed to place pebbles on as many nodes as it wants, subject to the condition that their immediate predecessors already have pebbles, and to remove any pebbles it no longer wants. The Cumulative *Pebbling* Complexity (CPC), or cumulative space, of this computation is defined as the sum of the number of pebbles on the graph after each step. Because pROM computation can simply follow the pebbling game, $\text{CMC}(F) \leq w \cdot \text{CPC}(G)$ (ignoring memory used for bookeeping data structures), where w is the output length of RO. As shown by [AS15], the converse is also essentially true (subject some bookkeeping factors): for data-independent functions (iMHFs), every pROM computation essentially follows the pebbling game, and $\text{CMC}(F)$ is roughly $\text{CPC}(G)/w$. Thus it is sufficient to simply analyze the pebbling game, because implies high CMC. In other words, storing labels in some other form than atomically does not improve the CMC for iMHFs.

DAG-based dMHFs The question that has been open for some time, and that we address in this paper, is whether the same holds for data-*dependent* memory-hard functions (dMHFs). Because dMHFs are not representable as static DAGs, we have to first describe the class of dMHFs F we consider, so that a pebbling game is meaningful for a function F. We start by recalling the first MHF construction scrypt, which can be viewed as a dynamic DAG. In such a DAG, the random oracle is used for two distinct purposes: not only for the computation of the labels, but also for building the DAG itself as the labeling computation progresses. Namely, the origins of incoming edges to a node u become known only when the previous node $u-1$ is labeled with some label ℓ_{u-1}: they are given by some function of ℓ_{u-1}. In scrypt, this idea is implemented essentially as follows: the first half of the n-node graph is simply a static line, and thus for every u, $\ell_u = RO(u_{\ell-1}, u)$; in the second half of the graph, every node u gets an additional incoming edge sampled uniformly from the first half, given by $\ell_{u-1} \bmod n/2$. Thus, in the second half, $\ell_u = RO(u_{\ell-1}, u_{(u_{\ell-1} \bmod n/2)}, u)$. Effectively, the incoming edges are sampled at random as the second half of the computation progresses.

The Randomized Pebbling Game for Pure Pebblers DAG-based dMHFs were formalized by [ACKKPT16b, Section 2.1] as a randomized pebbling game pebble: the game is specified by a DAG G (which is not dynamic) and a query set $\mathcal{Q}$ of nodes of G. A player is asked repeatedly, t times, to play the parallel pebbling game to reach nodes of G picked uniformly from $\mathcal{Q}$ (it gets to keep, add, and remove pebbles between queries according to the same rules as before). The CPC (called cc in [ACKKPT16b]) of this game is defined as before: it is the expected sum of the number of pebbles used at each pROM step, for all the steps needed to answer t queries to G. We make one slight change to allow for some delay δ between queries (the player thus gets to decide whether to incur the cost of keeping pebbles on G during this time or risk erasing them and incurring the cost of needing to repebble G). In the graph view of dMHF, this delay corresponds to having a line of δ degree-1 nodes before the next degree-2 node.

Entangled Pebbling To model XOR operations of an entangled pebbler on a DAG-based dMHF, we allow the following additional operations during the pebbling game. Every set of d nodes may have an entangled pebble on it. The rule for placing it is that all d nodes have to have a pure pebble on them. The rule for using it is that the posession of an entangled pebble for d nodes and pure pebbles for $d-1$ of them entitles the pebbler to place a pebble on the remaining node. Entangled pebbles cost the same amount of space as pure ones, but no time is charged for placing them and using them. (More sophisticated entangled pebbles and operations are considered in [ACKKPT16b] but are not needed to get our separation.)

Definition 2.2. A function F is a DAG-based dMHF if its computation is representable by the pebbling game as described here, with a graph G, t queries from $\mathcal{Q}$, and delays δ between queries. We denote by $\mathrm{CPC}(F)$ and $\mathrm{CPC}^{\updownarrow}(F)$ the CPC of the best pure pebbler and entangled pebbler, respectively. The *size* n of F refers to the minimum total number of RO queries required to compute F (equivalently, number of nodes in the dynamic graph).

2.3 Ingredient: Butterfly graphs

Let n be a power of two. The *butterfly graph* B_n (see Figure 1) is a layered DAG on with $1+\log n$ layers of n vertices each. The vertices are labeled according to the layer and their position within the layer, i.e., ,$v_{\ell,x}$, $\ell \in \{0,1,\ldots,\log n\}$, $x \in \{0,\ldots,n-1\}$. Edges go from each layer to the one

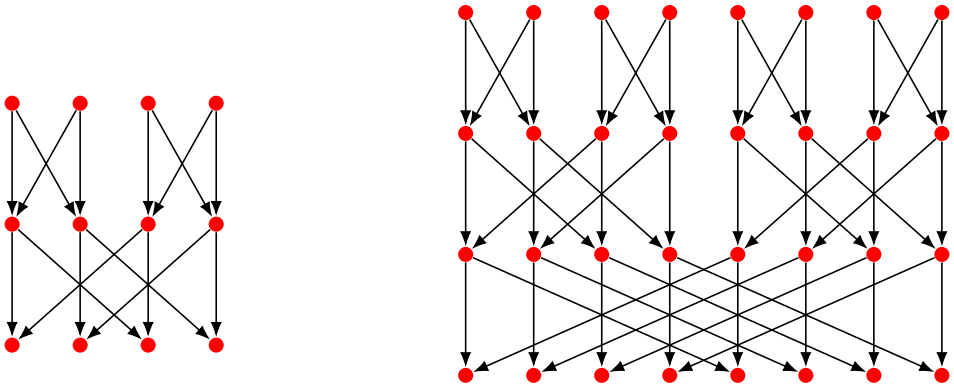

Figure 1: Examples of butterfly graphs, B_4 (left) and B_8 (right)

below it, with outdegree two (except for the last layer $\log n$, which is all sinks), according to the following rule: the edge $(v_{\ell,x}, v_{\ell+1,x'})$ exists if $x = x'$ or x and x' differ only in the ℓth bit.

Note that there is a unique path from every source $v_{0,x}$ to every sink $v_{\log n,y}$, in which each edge either keeps the corresponding bit in the binary representation or changes it, depending on whether x and y differ in that bit position. Note also that every node has $v_{\ell,z}$ has n source-to-sink paths passing through it: namely the paths from $v_{1,x}$ to $v_{\ell,y}$ for every x that agrees with z on the last $\log n - \ell$ bits and every y that agrees with z on the first ℓ bits.

We will also consider source-truncated (respectively, sink-truncated) butterfly graphs, in which some number of nodes in layer 0 and their outgoing edges (respectively, some number of nodes in layer $\log n$ and their incoming edges) are removed.

3 Main Construction Used for Both Separations

All our constructions start with an existing proof of space $\hat{T}$. Assume it has $\hat{t}_{\text{init}}$ total nodes (and thus $\hat{t}_{\text{init}}$ initialization cost) of which $\hat{s}$ are output nodes $\hat{Y}$, and achieves space ratio $\hat{r}_{\text{space}}$, hardness ratio $\hat{r}_{\text{hardness}}$, and $\hat{q}$-query soundness error $\hat{\sigma}$. We assume that queries in $\hat{T}$ are simply random subsets of $\hat{Y}$ of size $\hat{q}$. For both PoS and dMHF separations, we build a new PoS T out of $\hat{T}$.

3.1 The New Proof-Of-Space Construction

Warm-up: Technical Approach for Saving Half the Storage We first describe a simple version T_{simple} of our construction.

Construction 3.1. Given a parameter k—which will determine the speed-up of the entangled pebbler vs. the pure pebbler—and given a PoS $\hat{T}$ with $\hat{s} = k$ output nodes, we construct a PoS $(T_{\text{simple}}, \mathcal{Q})$ as follows (see Figure 2). T_{simple} consists of replicating $\hat{T}$ a total of $k+1$ times to get $\hat{T}_1, \ldots, \hat{T}_{k+1}$, where $k = \hat{s}$ (and joining the $k+1$ source nodes together into one). Let $\hat{Y}_i$ be the set of $\hat{s}$ outputs of $\hat{T}_i$. The set of output vertices of T_{simple} is $Y = \bigcup_i \hat{Y}_i$.

In order to define $\mathcal{Q}$, we index the nodes of $\hat{Y}_j$ with indices in $[k+1] - \{j\}$ so that $\hat{Y}_j = \{\hat{y}_{ji} : i \in [k+1] - \{j\}\}$. That is, $\hat{T}_j$ has one output node corresponding to every other $\hat{T}_i$. To make a query, we pick a random i and request the single node that corresponds to i from every $\hat{T}_j$ for

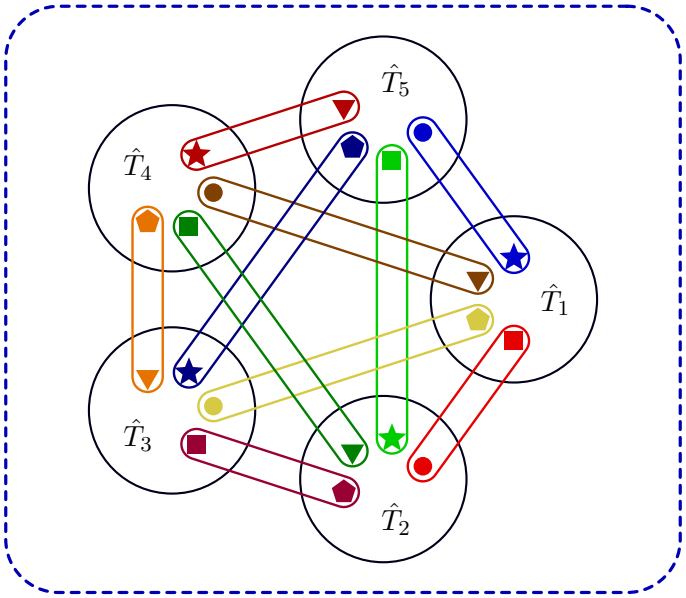

Figure 2: PoS T_{simple} from Construction 3.1. Each circle represents a $\hat{T}_i$; only the nodes $\hat{y}_{ij}$ in each $\hat{Y}_i$ are shown and are depicted as different shapes. Each of the 5 shapes represents a query Q_i (of 4 nodes each). Each color represents an entangled pebble.

$j \neq i$. Thus, the set $\mathcal{Q}$ consists of Q_i for $i \in [k+1]$ defined as $Q_i = \{\hat{y}_{ji} : j \in [k+1] - \{i\}\}$. (Such a query is big—of size k—but we will fix this problem with Construction 3.2.)

An honest pure pebbler will simply store all the $\hat{y}_{ij}$ values for every ordered pair (i, j) with $i \neq j$. A dishonest pure pebbler can try to store less than $r_{\text{space}} < \hat{r}_{\text{space}}$ fraction of the nodes of all $\hat{Y}_i$s; however, for any $\hat{Y}_i$ for which the stored fraction is below $\hat{r}_{\text{space}}$, work will be $\hat{r}_{\text{hardness}}\hat{t}_{\text{init}}$ with probability $1 - \hat{\sigma}$ if we repeat random queries $\hat{q}$ times. In other words, a dishonest pure pebbler who tries to save some fraction of the storage space beyond $\hat{r}_{\text{space}}$ will pay a proportional fraction of the initialization cost.

On the other hand, an adversary who is allowed to entangle pebbles will simply store XORs of all pairs of labels: $\hat{y}_{ij} \oplus \hat{y}_{ji}$ for every $i \neq j$. In response to a verification query, this adversary will need to reconstruct only $\hat{Y}_i$, at the cost of only $1/(k+1)$ of total initialization cost; this will give all the $\hat{y}_{ij}$ values, and computing XORs with stored entangled pebbles will give the $\hat{y}_{ji}$ values that are needed to respond to the query. This adversary will save half the space and will pay only a $\frac{1}{k+1}$ fraction of the initialization cost for each query.

We now show how to go from the highly structured and large queries to ones that are uniformly random and small.

Construction 3.2. Assume k is a power of 2. Take Construction 3.1 and add $k+1$ butterfly graphs B_k, which we call *query connectors*, and identify the sources of the ith butterfly graph with the vertices in Q_i (see Figure 3). For each i, such a graph has $\log k$ layers of k nodes each, with the source layer being the $\hat{y}_{ji}$ nodes (one node from each $\hat{Y}_j$ for $j \neq i$).

In a query connector, a single sink node is connected to all k source nodes, and thus, if there are no pebbles on the graph, a query of size 1 can replace a query of size k. Consider now the

case when all the pebbles are on the source nodes of the query connectors (i.e., on the $\hat{Y}_j$s). Then a single query to a random sink node hits all the unpebbled sources, and is thus equivalent to k queries $\hat{y}_{ji}$. Of course, pebbles do not have to be on the source nodes; however, because of the butterfly structure, if there are p pebbles on the graph, then a random sink node is connected, in expectation to $k - p$, regardless of how the pebbles are positioned.

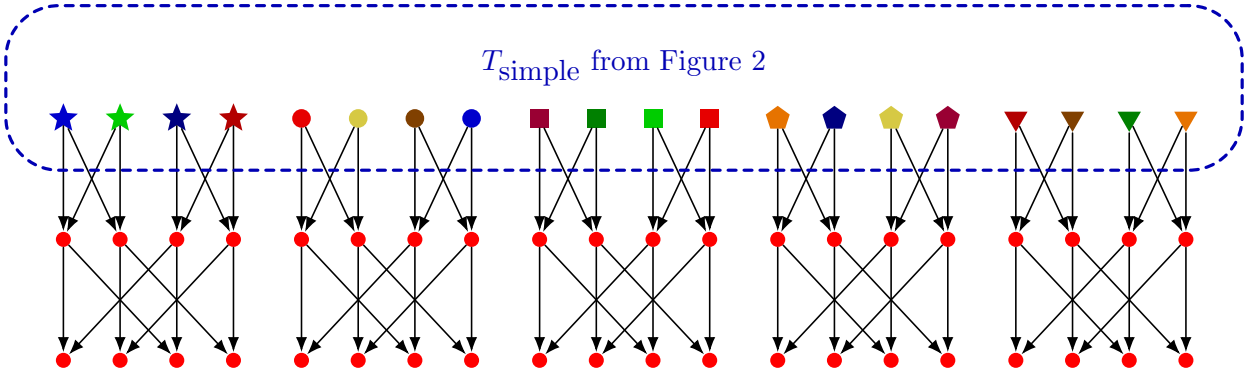

Figure 3: PoS from Construction 3.2. The box contains the entire graph T_{simple} from Figure 2; only its output nodes are shown.

Technical Approach for Saving More Storage We now want to allow the entangled pebbler to save a factor of d of storage (instead of a factor 2 presented above). We thus generalize from pairs $(\hat{y}_{ij}, \hat{y}_{ji})$ to sets of size d, as follows. Let the number of copies be $c = (k+1)(d-1)$. Figure 4 helps explain the construction. For $d = 2$, it is the same as Construction 3.1.

Construction 3.3. Given parameters k and d—which will determine the speed-up and space savings, respectively, of the entangled pebbler vs. the pure pebbler—and a PoS $\hat{T}$ with $\hat{s} = \binom{c-1}{d-1}$ output nodes, where $c = (k+1)(d-1)$, we construct a PoS $(T, \mathcal{Q})$ as follows. The DAG T consists of replicating $\hat{T}$ a total of c times to get $\hat{T}_1, \ldots, \hat{T}_c$, where $k = \hat{s}$ (and joining their source nodes together into one). The set of output vertices of T is $Y = \bigcup_i \hat{Y}_i$.

In order to define $\mathcal{Q}$, we index the nodes of $\hat{Y}_j$ with distinct size-$(d-1)$ subsets of $[c] - \{j\}$ (that is the reason each $\hat{Y}_j$ is of size $\hat{s} = \binom{c-1}{d-1}$). In other words, $\hat{Y}_j$ contains one vertex $\hat{y}_{jU}$ for every size-$(d-1)$ subset $U \subset [c] - \{j\}$. To make a query, we pick a random size-$(d-1)$ subset $U \subset [c]$ and request the single node $\hat{y}_{jU}$ corresponding to U from every $\hat{Y}_j$ for $j \notin U$. Let $Q_U = \{\hat{y}_{jU} : j \notin U\}$ and $\mathcal{Q} = \{Q_U : U \subset [c], |U| = d-1\}$. Note that $|Q_U| = c - (d-1) = k(d-1)$.

Again, we can add butterfly graphs to reduce the query size and remove the need for complex query structure.

Construction 3.4. Given parameters q, k and d such that $k(d-1)$ is a power of 2, we build a PoS $(T, \mathcal{Q})$ as follows. Start with the PoS built in Construction 3.3 and, for each of its queries Q_U, add a butterfly graph with sources Q_U. Each butterfly graph thus adds $\log k(d-1)$ layers of size $k(d-1)$ each. We call such butterfly graphs *query connectors*. Let T be the resulting graph. The sinks of the query connectors are the output nodes Y (the output size remains $s = c \cdot \binom{c-1}{d-1}$). A query of size q is constructed as follows: choose q query connectors and one node in each; the set of all such choices constitutes $\mathcal{Q}$.

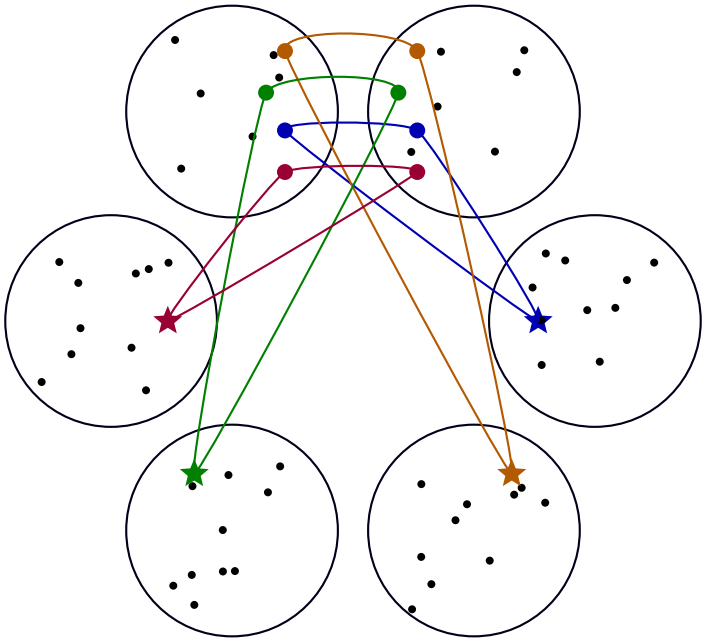

Figure 4: PoS from Construction 3.3 for $k = 2$, $d = 3$. Each of the $(2+1)(3-1) = 6$ circles represents $\hat{T}$ with $|\hat{Y}|$ of size $\binom{5}{2} = 10$. Nodes corresponding to a single query Q_U of size 4 are shown as stars. Each entangled pebble used to answer Q_U is shown as a 3-hyperedge.

The calculation for the tradeoff between storage and work for an adversarial pure pebbler does not change much with the generalization for $d > 2$ and with the addition of query connectors: reducing storage by r_{space}-fraction will cost roughly $(1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}}})\hat{r}_{\text{hardness}} t_{\text{init}}$, though there are smaller terms to account for (see Theorem 4.3).

The entangled pebbler, however, can gain a strong advantage by entangling sets of pebbles of size d via the following strategy.

Strategy 3.5. For each $S \subset [c]$ of size d we define the set

$$E_S = \{\text{the vertex } \hat{y}_{jU} \in \hat{Y}_j \text{ corresponding to } U = S - \{j\} \mid j \in S\}. \tag{1}$$

Note that the sets E_S partition Y (because every vertex in Y is indexed as $\hat{y}_{jU}$ and thus belongs to E_S for $S = U \cup \{j\}$). An entangled pebbler can now store an entangled pebble for each set E_S, requiring s/d pebbles in total. When a query to a particular query connector is issued, its sources form a query Q_U that corresponds to a size-$(d-1)$ set U. The entangled pebbler will pebble $\hat{T}_i$ fully for each $i \in U$. These computed pebbles plus the stored entangled pebbles can be used to immediately pebble every vertex in Q_U. This requires $d-1$ copies of $\hat{T}$ to be pebbled, as compared to the $|Q_U| = k(d-1)$ copies required for a pebbler not storing anything. Given the pebbles on the nodes of Q_U, it is then easy to pebble the actual queried sink of the query connector: it requires simply pebbling a portion of the butterfly graph that is a binary tree whose roots are known, and thus takes work $k(d-1) < c$, which is a small fraction of t_{init}.

3.2 Adaptation to dMHFs

To build a DAG-based dMHF that separates pure pebblers from entangled ones, we use Construction 3.4 as the graph G for the randomized pebbling game described in Section 2.2. The dMHF thus consists of this G followed by a dynamic graph that is a line with an incoming edge from a random sink of G every δ steps (see Figure 6). The randomized pebbling game, or, equivalently, random edges in a dynamic graph, simulate PoS verifier queries, and thus, intuitively, should give the same separation.

One difference between the PoS construction and the dMHF construction, however, is that PoS complexity is measured in sequential time, whilst we now wish to measure CPC. Both are additive over the c copies of $\hat{T}_i$ graphs, so we just need the underlying graph $\hat{T}$ whose CPC does not drop much after some pebbles are stored on it. The second difference is that in this setting we care about expected work, rather than guaranteed work, but that only simplifies the arguments. Thus, our goal is to find a suitable $\hat{T}$.

We need $\hat{T}$ to be a good proof of space in the expected CPC setting. More formally let $\hat{T}$ be a DAG with a distinguished set of vertices $\hat{Y}$ of size $\hat{s}$. Consider the following game between a pebbler and a verifier. The pebbler may start with any configuration of pebbles. The verifier selects a vertex in $\hat{Y}$ uniformly at random, which the pebbler then needs to pebble in low cumulative space. We say that $\hat{T}$ achieves space and hardness ratios $\hat{r}_{\text{space}}$ and $\hat{r}_{\text{hardness}}$, if any pebbler starting with a configuration containing at most $\hat{r}_{\text{space}}\hat{s}$ pebbles has expected CPC at least $\hat{r}_{\text{hardness}} \cdot \widehat{cpc}_{\text{init}}$, where $\widehat{cpc}_{\text{init}}$ is the CPC of a pebbler who starts with pebbles only on the source nodes.

To achieve the strongest possible separation we need both $\hat{r}_{\text{space}}$ and $\hat{r}_{\text{hardness}}$ to be as large as possible, ideally both constant. This is a problem that sits in between PoSes and MHFs, and has not been studied in either literature. Moreover, we need this graph to have relatively low depth, in order for the entangled pebbler to have low CPC, while typical research on MHFs looks for high-depth graphs. Our saving grace is that we do not need $\hat{T}$ to have a high CPC; it suffices that it is robust to initial pebbles being placed on the graph, which makes the problem somewhat easier. We achieve this with the following simple construction.

Construction 3.6. Given a positive integer $\hat{s}$, we construct a DAG $\hat{T}$ as follows (see Figure 5). Let $\hat{s}' = 2^{\lceil \log \hat{s} \rceil}$ be the smallest power of 2 at least as large as $\hat{s}$. Take the butterfly graph $B_{\hat{s}'}$, which we call B, create a new source node, and prepend a line of length $\log \hat{s}'$ from the new source node to each source of B, so that the total depth is $2 \log \hat{s}'$. If $\hat{s}$ is not a power of 2, we use a sink-truncated version of $B_{\hat{s}'}$, so that exactly $\hat{s}$ sinks remain.

Note that the naive strategy, pebbling one layer at a time, pebbles the last layer in parallel time $2 \log \hat{s}'$ and CPC at most $2\hat{s}' \log \hat{s}'$, i.e., linear in the size of the graph. For the rest of the section, $\hat{T}$ will refer to this graph.

Claim 3.7. *Let $\hat{T}$ be given by Construction 3.6. Any pure pebbling strategy storing p pebbles will in expectation need at least $(\hat{s} - p) \log \hat{s}$ cumulative space to answer a random challenge to the last layer of $\hat{T}$.*

Proof. Say that the deterministic pebbler stores p_1 pebbles on the lines above B and p_2 pebbles on B. Let v be a sink in B and H_v be the subset of sources in B with an unpebbled path to v. There is a path from every source to every sink in B; furthermore every pebble blocks at most $\hat{s}'$ such paths. This means that $\hat{s}'(\hat{s} - p_2)$ such paths are unpebbled and therefore the expected size of H_v (over a random choice of v) is $\frac{\hat{s}'}{\hat{s}}(\hat{s} - p_2) \geq \hat{s} - p_2$.

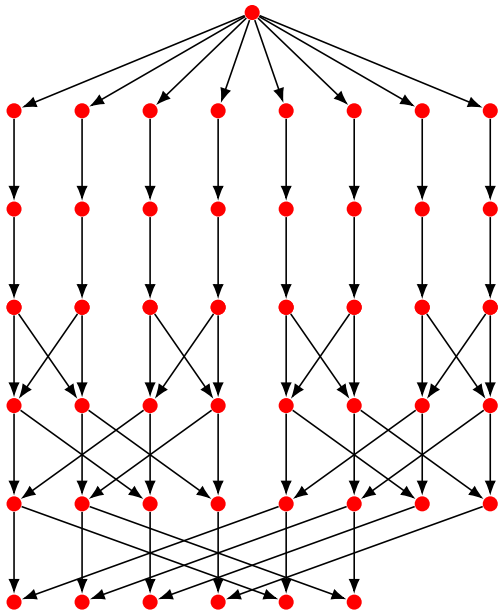

Figure 5: $\hat{T}$ from Construction 3.6 using $\hat{s} = 6$.

Now consider the cumulative space required to pebble H_v. Consider the line above some node $u \in H_v$. At least $H_v - p_1$ such lines have no pebbles in them, and to pebble those lines will take CPC of $\log \hat{s}' \leq \log \hat{s}$ per line. This gives us an expected CPC of at least

$$\mathop{\mathbb{E}}_{v}[(|H_v| - p_1) \log \hat{s}] = (\mathbb{E}[|H_v|] - p_1) \log \hat{s} \geq (\hat{s} - p_1 - p_2) \log \hat{s} = (\hat{s} - p) \log \hat{s}\,. \qquad \square$$

Given this $\hat{T}$, we present our full dMHF construction.

Construction 3.8. Given a parameter k, which will determine the size of the construction, we construct a dMHF as follows (see Figure 6). Let $d = k$, $c = (k+1)(d-1)$, and $\hat{s} = \binom{c-1}{d-1}$. Apply Construction 3.4 to the graph $\hat{T}$ of Construction 3.6 to create a graph T; let $G = T$. Note that Construction 3.4 requires $k(d-1)$ to be a power of 2, which is not possible here; thus, we use a source-truncated butterfly graph for each query connector. Let L be a line graph consisting of s segments of length $\delta = \lceil \log \hat{s} \rceil$. For every vertex v at the end of each segment, add a (data dependent) edge from a random sink of G to v.

Finally, we note that the graph G can be pebbled from the source to the sinks in CPC linear in the size of the graph. This follows by the naive strategy that pebbles one layer at a time in parallel. We record this observation for future reference.

Observation 3.9. *The graph G from Construction 3.8 can be pebbled in* CPC *linear in the size of the graph. Moreover, this pebbling can be done so that at the last step all sinks are pebbled.*

4 The Pebbling vs Random Oracle Separation for Proofs of Space

In this section we formally state and prove our result for Proofs of Space.

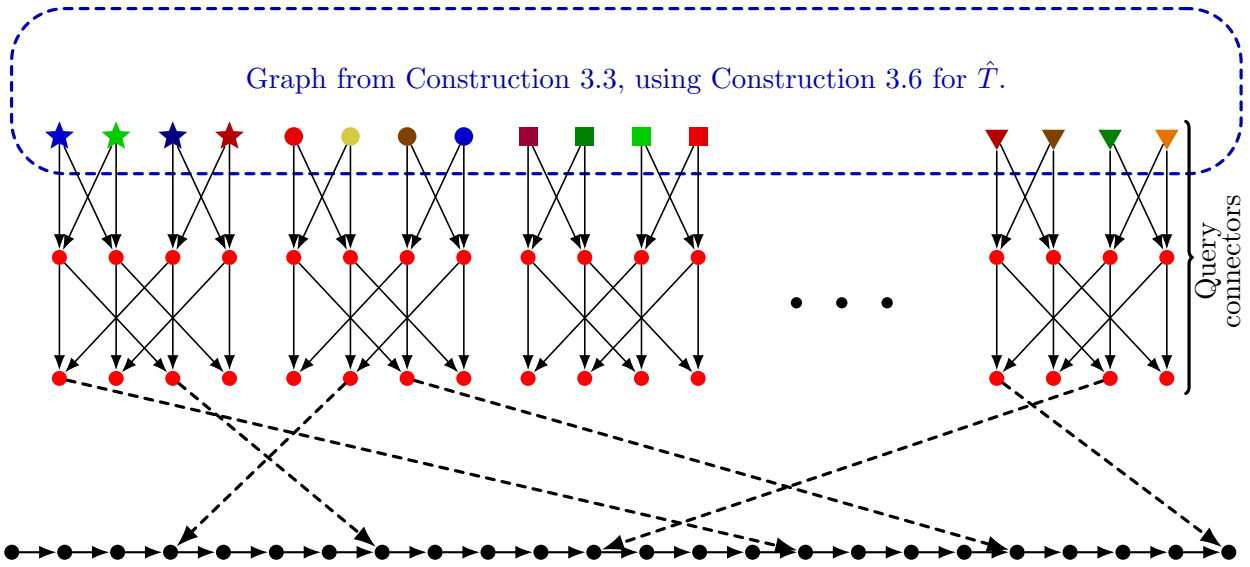

Figure 6: dMHF of Construction 3.8. Everything but the last line forms the graph G. Dashed arrows indicate dynamically discovered edges.

Theorem 4.1. *Let r_{space}, r_{hardness}, $\sigma \in (0,1)$ be such that $r_{\text{space}} + r_{\text{hardness}} < 1$. Then there exists a constant q such that for any positive integers $k \geq 3$, $d \geq 2$, there exists a PoS T (Construction 3.4) with challenges of size q that has the following properties:*

- *in the pure pebbling model, T achieves a space ratio, hardness ratio, and soundness error of r_{space}, r_{hardness}, and σ respectively,*
- *there exists an entangled pebbler that, storing $\frac{s}{d}$ pebbles, can answer challenges in time $\frac{qt_{\text{init}}}{k+1}$; in particular it is not possible to simultaneously achieve $r_{\text{space}} > \frac{1}{d}$ and $r_{\text{hardness}} > \frac{q}{k+1}$ in the random oracle model.*

We prove this theorem in three steps, and then refine it to evaluate the result for a concrete graph.

4.1 The Entangled Pebbler's Algorithm

Lemma 4.2 (upper bound)**.** *An entangled pebbler storing $\frac{s}{d}$ entangled pebbles can answer any challenge of size q in time at most $\frac{qt_{\text{init}}}{k+1}$.*

Proof. Let c be the number of copies of $\hat{T}$ given by Construction 3.4. For every size-d subset of $[c]$ the entangled pebbler stores an entangled pebble of the d corresponding vertices.

Consider first a challenge to a single vertex. Such a query corresponds to a set Q_U in Construction 3.3. The entangled pebbler can pebble such a set by pebbling the set of $d-1$ $\hat{T}_i$s for $i \in U$ and then using the entangled pebbles to pebble Q_U. This requires the entangled pebbler to pebble a $\frac{1}{k+1}$ fraction of $\hat{T}_i$s. Furthermore the entangled pebbler needs to pebble one of the $\binom{c}{d-1} \geq k+1$ query connectors. Thus the entangled pebbler needs at most $\frac{t_{\text{init}}}{k+1}$ work per query. □

4.2 Lowerbounding the Work of the Pure Pebbler

To show that we can achieve good hardness in the pebble model we first prove the following more general theorem, showing how the hardness of T depends on the properties of $\hat{T}$.

Theorem 4.3 (Lower bound)**.** *Suppose $\hat{T}$ is a proof of space with output $\hat{Y}$ of size $\hat{s}$, space ratio $\hat{r}_{\text{space}}$, hardness ratio $\hat{r}_{\text{hardness}}$, and soundness error $\hat{\sigma}$ after a random subset of $\hat{Y}$ of size $\hat{q}$ is queried. Set $\varepsilon = \sqrt{\frac{\ln 1/\hat{\sigma}}{2\hat{s}}}$.*

Then Construction 3.4 for any $k \geq 4$ and $d \geq 2$, such that $k(d-1)$ is a power of 2, is a proof of space in the pure pebbling model (per Definition 2.1) that, for any soundness error $\sigma > 3\hat{\sigma}$ and any $r_{\text{space}} > \hat{r}_{\text{space}} - \varepsilon$, has hardness ratio

$$r_{\text{hardness}} \geq \left(1 - \frac{3\hat{\sigma}}{\sigma}\right)\left(1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}} - \varepsilon}\right) \cdot \frac{\hat{t}_{\text{init}}}{\hat{t}_{\text{init}} + \hat{s}\log(k(d-1))} \cdot \hat{r}_{\text{hardness}}$$

when a query Q selects a random sink in each of the q randomly chosen query connectors, for any q satisfying $\exp(-2q(k/(k+1) - \hat{q}/q)^2) < \hat{\sigma}$ (in particular, it suffices that $q \geq \max(4\hat{q}, 2\ln(1/\hat{\sigma}))$.

We note that the choice of ε and q is somewhat arbitrary; bigger ε and q can reduce $3\hat{\sigma}$ in the first multiplicand, but will make the formulas harder to understand. We use a bigger ε and eliminate the dependence on q in Section 4.4.

Proof. Fix a pebbling that stores at most $r_{\text{space}}s$ total pebbles.

To analyze a query set that consists of q random challenges to distinct query connectors, we think of the the process of choosing q sinks of distinct query connectors as two separate steps. First we sample a vector $\mathbf{v} \sim [k(d-1)]^{\binom{c}{d-1}}$ that for each query connector selects one sink. Then we sample a random subset $\mathbf{J}$ of q query connectors. The selected sink of each selected query connector graph gives the set of q challenges.

Consider one of the small proofs of space, $\hat{T}_i$, for a fixed i. Every vertex $u \in \hat{Y}_i$ is a source vertex in exactly one query connector. Let p_u be the probability that a random sink in this query connector has a pebble on its unique path from u (in particular, a pebble on u makes $p_u = 1$). We say that u has a *fractional pebble of weight p_u*. Let f_i be the actual number of pebbles on $\hat{T}_i - \hat{Y}_i$ plus $\sum_{u \in \hat{Y}_i} p_u$.

Lemma 4.4. *$\sum_i \sum_{u \in \hat{Y}_i} p_u$ is at most the number of pebbles in the query connectors (including those on $\hat{Y}_i$) and thus $\sum_i f_i$ is at most the total number of pebbles, $r_{\text{space}}s$.*

Proof. Recall that a query connector is a butterfly graph with n nodes (in our case, $n = c-(d-1)$). It has n^2 source-to-sink paths, with every source and sink having a distinct path between them. Every pebble blocks at most n paths (because there are n paths through every node), so if there are p pebbles, there are at least $n^2 - pn$ unblocked paths remaining. The probability $(1-p_u)$ that a particular source is connected to a randomly chosen sink is equal to the number of paths that start at this source and go to any sink, divided by n. Thus, the sum of these probabilities $(1-p_u)$ over all n sources u of a single butterfly graph is equal to the total number of unblocked source-to-sink paths, divided by n; i.e., $(n^2 - np)/n = n - p$. So the sum of p_u for every source u in a single butterfly graph is at most the number of pebbles on the graph. The result follows by adding up over all the query connectors. □

For every node $u \in \hat{Y}_i$, fixing the sink of the query connector to which it belongs is equivalent to flipping a p_u-biased coin in the following sense: with probability p_u, u has a path to this sink and thus has a chance of being needed to answer a query, and with probability $1 - p_u$ it has no such path and thus will not be needed to answer a query. Thus, after $\mathbf{v}$ is fixed, we can consider a new, *virtual* set of pebbles, in which we remove all the pebbles from the query connectors and add a pebble to every node u in every $\hat{Y}_i$ that has no path to $\mathbf{v}$. In expectation, this virtual set of pebbles has as many pebbles as the actual set of pebbles, because we remove $\sum_i \sum_{u \in \hat{Y}_i} p_u$ pebbles (per Lemma 4.4) and add, on average, the same number of virtual pebbles. Thus, letting $p_i(\mathbf{v})$ be the number of virtual pebbles in $\hat{T}_i$ after $\mathbf{v}$ is fixed, we know that the expectation, over the choice of $\mathbf{v}$, of $p_i(\mathbf{v})$, is f_i.

Now, let $\hat{T}_i$ be called "costly" if $f_i \leq (\hat{r}_{\text{space}} - \varepsilon)\hat{s}$. For such $\hat{T}_i$, $p_i(\mathbf{v})$ will likely not exceed $\hat{s}\hat{r}_{\text{space}}$ (which is what we need to argue that the $\hat{T}_i$ may contribute to pebbler's work). Specifically, we can use Hoeffding's inequality (which applies because every element of $\mathbf{v}$ is independent, and thus the coinflip for every node $u \in \hat{Y}_i$ is independent, made by the element of $\mathbf{v}$ that corresponds to its query connector),

$$\Pr_{\mathbf{v}}[p_i(\mathbf{v}) > \hat{s}\hat{r}_{\text{space}}] \leq \exp(-2\varepsilon^2 \hat{s}) \leq \hat{\sigma}\,, \tag{2}$$

where the second step follows by the definition of ε from the theorem statement,

Recall that $\mathbf{J}$ determines which q query connectors get queried. Once $\mathbf{v}$, and thus the virtual pebbling, are fixed, every single query to a query connector corresponds to a query Q_U from Construction 3.3: namely it chooses a single vertex from all but $d-1$ of the $c = (k+1)(d-1)$ $\hat{T}_i$s. Let $\mathbf{C}_i(\mathbf{J})$ be the subset of $\hat{Y}_i$ that belongs to the query connectors chosen by $\mathbf{J}$. The expected size of $\mathbf{C}_i(\mathbf{J})$ is $\lambda \cdot \frac{k}{k+1}$. To have $\hat{T}_i$ cause pebbler work with probability $1 - \hat{\sigma}(\hat{q})$, we need $|\mathbf{C}_i(\mathbf{J})| \geq \hat{q}$. By Hoeffding's inequality,

$$\begin{aligned}
\Pr_{\mathbf{C}_i(\mathbf{J}}[|\mathbf{C}_i(\mathbf{J})| < \hat{q}] &\leq \exp\left(-2\left(q \cdot \frac{k}{k+1} - \hat{q}\right)^2 / q\right) \\
&= \exp\left(-2q\left(\frac{k}{k+1} - \frac{\hat{q}}{q}\right)^2\right) \\
&\leq \hat{\sigma}, \qquad (3)
\end{aligned}$$

where where the last line is true because of the condition on q in the theorem statement.

For each costly $\hat{T}_i$, let $\mathbf{B}_i(\mathbf{v}, \mathbf{J})$ be the bad event that the pure pebbler does not require $\hat{r}_{\text{hardness}}\hat{t}_{\text{init}}$ work to pebble the queried vertices in $\hat{T}_i$. Note that this event only depends on $\mathbf{v}$ and $\mathbf{J}$. We have that

$$\begin{aligned}
\Pr_{\mathbf{v},\mathbf{J}}[\mathbf{B}_i(\mathbf{v}, \mathbf{J})] \leq &\Pr_{\mathbf{v}}[p_i(\mathbf{v}) > \hat{s}\hat{r}_{\text{space}}] + \Pr_{\mathbf{J}}[|\mathbf{C}_i(\mathbf{J})| < \hat{q}] \\
&+ \Pr_{\mathbf{v},\mathbf{J}}[\mathbf{B}_i(\mathbf{v}, \mathbf{J}) \mid p_i(\mathbf{v}) \leq \hat{s}\hat{r}_{\text{space}}, |\mathbf{C}_i(\mathbf{J})| \geq \hat{q}] \leq 3\hat{\sigma}\,,
\end{aligned}$$

where the last inequality follows per equations (2) and (3), and because the last term is, by definition, $\hat{\sigma}$, because $\mathbf{C}_i(\mathbf{J})$ is a uniform query of size at least $\hat{q}$ to $\hat{Y}_i$.

Thus, for each costly $\hat{T}_i$, the probability of not requiring $\hat{r}_{\text{hardness}}\hat{t}_{\text{init}}$ work is at most $3 \cdot \hat{\sigma}$.

Letting I be the set of indices i such that $\hat{T}_i$ is costly, we have

$$\Pr_{\mathbf{v},\mathbf{J}}\left[\sum_{i\in I}\mathbf{B}_i(\mathbf{v},\mathbf{J}) > \frac{3\cdot\hat{\sigma}\cdot|I|}{\sigma}\right] \leq \sigma \tag{4}$$

by Markov's inequality, since $\mathbb{E}_{\mathbf{v},\mathbf{J}}[\sum_{i\in I}\mathbf{B}_i(\mathbf{v},\mathbf{J})] = \sum_{i\in I}\mathbb{E}_{\mathbf{v},\mathbf{J}}[\mathbf{B}_i(\mathbf{v},\mathbf{J})] \leq |I|\cdot 3\cdot\hat{\sigma}$.

Note that out of the c total proofs of space $\hat{T}_i$, at least

$$|I| \geq \left(1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}} - \varepsilon}\right)c \tag{5}$$

are costly (since if fewer than that are costly, then by definition of costly, $\sum_i f_i > \frac{r_{\text{space}}}{\hat{r}_{\text{space}}-\varepsilon}c\cdot(\hat{r}_{\text{space}} - \varepsilon)\hat{s} = r_{\text{space}}s$, but by Lemma 4.4, $\sum_i f_i \leq r_{\text{space}}s$). Since the costs of answering the queries for each $\hat{s}_i$ are additive (since their nodes do not overlap), and with probability $1-\sigma$, there are $\left(1-\frac{3\hat{\sigma}}{\sigma}\right)|I|$ costly ones that actually cost $\hat{r}_{\text{hardness}}\hat{t}_{\text{init}}$ to answer, the total cost, with probability at least $1-\sigma$, is at least

$$\left(1-\frac{3\hat{\sigma}}{\sigma}\right)\left(1-\frac{r_{\text{space}}}{\hat{r}_{\text{space}}-\varepsilon}\right)c\cdot\hat{r}_{\text{hardness}}\hat{t}_{\text{init}} =$$
$$\left(1-\frac{3\hat{\sigma}}{\sigma}\right)\left(1-\frac{r_{\text{space}}}{\hat{r}_{\text{space}}-\varepsilon}\right)\cdot\frac{\hat{t}_{\text{init}}}{\hat{t}_{\text{init}}+\hat{s}\log k(d-1)}\cdot\hat{r}_{\text{hardness}}\cdot t_{\text{init}}\,,$$

because

$$t_{\text{init}} = c(\hat{t}_{\text{init}} + \hat{s}\log k(d-1)) \tag{6}$$

due to $c\hat{s}\log k(d-1)$ buttefly graph nodes. This concludes the proof of Theorem 4.3. □

4.3 Proof of Theorem 4.1

Now that we have proven an upper and lower bound, all that remains to prove Theorem 4.1 is finding $\hat{T}$ with appropriate parameters. We start by recalling Theorem 4.1 for convenience.

Theorem 4.1. *Let r_{space}, r_{hardness}, $\sigma \in (0,1)$ be such that $r_{\text{space}} + r_{\text{hardness}} < 1$. Then there exists a constant q such that for any positive integers $k \geq 3$, $d \geq 2$, there exists a PoS T (Construction 3.4) with challenges of size q that has the following properties:*

- *in the pure pebbling model, T achieves a space ratio, hardness ratio, and soundness error of r_{space}, r_{hardness}, and σ respectively,*
- *there exists an entangled pebbler that, storing $\frac{s}{d}$ pebbles, can answer challenges in time $\frac{qt_{\text{init}}}{k+1}$; in particular it is not possible to simultaneously achieve $r_{\text{space}} > \frac{1}{d}$ and $r_{\text{hardness}} > \frac{q}{k+1}$ in the random oracle model.*

Proof of Theorem 4.1. We wish to use Construction 3.4 to find the required T. Note that his construction only works when $k(d-1)$ is a power of 2. This is however not a problem since every construction that works for some choice of k and d is also a valid construction for any smaller k and d. It is thus sufficient to prove the theorem for k and d such that $k(d-1)$ is a power of 2, we will from now on assume that this is the case.

The second property of Theorem 4.1 now immediately follows from Lemma 4.2. Thus all we have to do is ensure the first property holds using using Theorem 4.3, which we do by finding a $\hat{T}$ and other parameters for which

$$\left(1 - \frac{3\hat{\sigma}}{\sigma}\right)\left(1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}} - \varepsilon}\right) \cdot \frac{\hat{t}_{\text{init}}}{\hat{t}_{\text{init}} + \hat{s}\log(k(d-1))} \cdot \hat{r}_{\text{hardness}}.$$

is greater than the desired r_{hardness}. Intuitively this expression consists of four multiplicands, that can all be set arbitrarily close to 1 (with the exception of $(1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}} - \varepsilon})$ that we instead set close to $1 - r_{\text{space}}$. All that remains to finish the proof is to fix parameters in the right order, making sure each factor is bounded close enough to one.

Let $\delta = 1 - \frac{r_{\text{hardness}}}{1 - r_{\text{space}}}$ (note that $\delta > 0$ because we assume $r_{\text{space}} + r_{\text{hardness}} < 1$). It suffices to ensure the following bounds hold for each of the four multiplicands that make up r_{hardness}:

$$\begin{aligned}
\left(1 - \frac{3\hat{\sigma}}{\sigma}\right) &\geq 1 - \frac{\delta}{4} \\
\left(1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}} - \varepsilon}\right) &\geq \left(1 - \frac{\delta}{4}\right)(1 - r_{\text{space}}) \\
\frac{\hat{t}_{\text{init}}}{\hat{t}_{\text{init}} + \hat{s}\log(k(d-1))} &\geq 1 - \frac{\delta}{4} \quad \text{and} \\
\hat{r}_{\text{hardness}} &\geq 1 - \frac{\delta}{4}.
\end{aligned}$$

We use [Rey24, Thm 1.] to find a PoS $\hat{T}$ achieving $\hat{r}_{\text{space}}$ and $\hat{r}_{\text{hardness}}$ arbitrarily close to 1. We demand $\hat{r}_{\text{hardness}} = 1 - \frac{\delta}{4}$ and pick

$$\hat{r}_{\text{space}} = 1 - \frac{2r_{\text{space}}}{\delta + 4r_{\text{space}} - \delta r_{\text{space}}}.$$

This construction has a 1-query soundness error of $\frac{1 + r_{\text{space}}}{2}$, i.e., constant. We amplify the soundness by selecting $\hat{q}$ large enough that $\hat{\sigma} < \frac{\sigma\delta}{12}$, yielding $1 - \frac{3\hat{\sigma}}{\sigma} < 1 - \frac{\delta}{4}$. We can now fix $q = \max(4\hat{q}, 2\ln(1/\hat{\sigma}))$.

We now need to choose $\hat{s}$ large enough so that

$$\varepsilon = \sqrt{\frac{\ln 1/\hat{\sigma}}{2\hat{s}}} \leq \frac{2r_{\text{space}}}{\delta + 4r_{\text{space}} - \delta r_{\text{space}}}. \tag{7}$$

This is in order to ensure that

$$1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}} - \varepsilon} \geq 1 - \frac{r_{\text{space}}}{1 - \frac{4r_{\text{space}}}{\delta + 4r_{\text{space}} - \delta r_{\text{space}}}} \geq \left(1 - \frac{\delta}{4}\right) \cdot (1 - r_{\text{space}}).$$

We can choose $\hat{s}$ satisfying (7) by choosing large enough $k' \geq k$ and $d' \geq d$, ensuring $k'(d'-1)$ is a power of 2, and setting $\hat{s} = \binom{c'-1}{d'-1}$ for $c' = (k'+1)(d'-1)$. In the end, we obtain a separation for k' and d', which implies the separation for k and d. We can therefore assume $k = k'$ and $d = d'$.

Now, given a fixed k and $\hat{r}_{\text{space}}$ (and thus a fixed $\hat{s}$) we wish to bound

$$\frac{\hat{t}_{\text{init}}}{\hat{t}_{\text{init}} + \hat{s}\log(k(d-1))}.$$

Increasing the number of layers ℓ in the SPR construction of [Rey24], increases $\hat{t}_{\text{init}}$ while keeping $\hat{s}$ constant and only improving the other parameters (this follows from [Rey24, Section VII]). Setting ℓ large enough thus ensures

$$\frac{\hat{t}_{\text{init}}}{\hat{t}_{\text{init}} + \hat{s}\log(k(d-1))} \geq 1 - \frac{\delta}{4}.$$

We have thus bounded all four multiplicands, concluding the Proof of Theorem 4.1.

□

4.4 Example Concrete Parameters

In this section, we demonstrate that for reasonable parameter values, the pure pebbler has to do 15 times more work than the entangled pebbler when using the same storage. To gauge reasonableness, we compare the parameters we obtain to those of the DAG-based PoS deployed at scale in the Filecoin cryptocurrency [Pro23].

To do so, we first modify Theorem 4.3 to measure expected hardness ratio ρ_{hardness} instead of measuring two parameters r_{hardness} and σ. This expected measure makes sense as a measure of the monetary cost of computation for a PoS adversary who chooses to reduce space. For this expected measure, it is natural to take $q = \hat{q} = 1$, because expected cost will grow at most linearly with q. We obtain the following theorem.

Theorem 4.5. *Suppose $\hat{T}$ is a proof of space with output $\hat{Y}$ of size $\hat{s}$, space ratio $\hat{r}_{\text{space}}$, hardness ratio $\hat{r}_{\text{hardness}}$, and soundness error $\hat{\sigma}$ after a single uniformly random node in $\hat{Y}$ is queried. Choose any $\sigma_\varepsilon > 0$ and let $\varepsilon = \sqrt{\frac{\ln 1/\sigma_\varepsilon}{2\hat{s}}}$.*

Then Construction 3.4 for any $k \geq 3$ and $d \geq 2$, such that $(k+1)(d-1)$ is a power of 2, is a proof of space in the pure pebbling model (per Definition 2.1) with expected hardness ratio

$$\rho_{\text{hardness}} \geq \frac{k}{k+1} \cdot \left(\left(1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}} - \varepsilon} \right) \cdot (1 - \hat{\sigma}) + (\hat{r}_{\text{space}} - \sigma_\varepsilon - r_{\text{space}}) \right) \times \frac{\hat{t}_{\text{init}}}{\hat{t}_{\text{init}} + \hat{s}\log k(d-1)} \cdot \hat{r}_{\text{hardness}} \cdot \tag{8}$$

Proof. We show only the changes from the proof of Theorem 4.3.

If $q = \hat{q} = 1$ and queries are just single random nodes, the proof of Theorem 4.3 can be simplified to remove the need for Equation (3). Instead, we observe that in the virtual pebbling, a random element of $\mathbf{J}$ has an unpebbled path to $\hat{Y}_i$ (if we don't count the possible pebbles on $\hat{Y}_i$ itself) with probability $\frac{k}{k+1}$. Because we are analyzing expected hardness, we do not need the Markov's inequality calculation in Equation (4). We thus obtain

$$\rho_{\text{hardness}} \geq \left((1 - \hat{\sigma}) \frac{k}{k+1} - \sigma_\varepsilon \right) \cdot \left(1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}} - \varepsilon} \right) \cdot \frac{\hat{t}_{\text{init}}}{\hat{t}_{\text{init}} + \hat{s}\log k(d-1)} \cdot \hat{r}_{\text{hardness}} \cdot$$

However, we can do even better by carefully estimating the contribution of every costly $\hat{T}_i$, because this contribution depends on the number of pebbles. We call a node in $\hat{Y}_i$ *costly* if computing it costs at least $\hat{r}_{\text{hardness}}\hat{t}_{\text{init}}$ work. We observe a continuous tradeoff between the number of virtual

pebbles and the number of costly nodes: if $p_i(\mathbf{v}) \le \hat{s}\hat{r}_{\text{space}}$, then $(1-\hat{\sigma}) + \left(\hat{r}_{\text{space}} - \frac{p_i(\mathbf{v})}{\hat{s}}\right)$ fraction of nodes in $\hat{Y}_i$ is costly.[2] Call this fraction c_i.

Observe now that for every costly i, a random element of $\mathbf{J}$ has an unpebbled path (with respect to the virtual pebbling) to a costly node in $\hat{Y}_i$ with probability $\frac{k}{k+1}c_i$. Thus, for a fixed $\mathbf{v}$ and random $\mathbf{J}$, the expected work to pebble $\hat{T}_i$ is at least $\frac{k}{k+1}c_i\hat{r}_{\text{hardness}}\hat{t}_{\text{init}}$. This gives us

$$\rho_{\text{hardness}} \ge \frac{k}{k+1} \cdot \left(\frac{1}{c}\sum_{i=1}^{c} \mathop{\mathbb{E}}_{\mathbf{v}} c_i\right) \cdot \frac{\hat{t}_{\text{init}}}{\hat{t}_{\text{init}} + \hat{s}\log k(d-1)} \cdot \hat{r}_{\text{hardness}}\,.$$

It remains to estimate $\sum_{i=1}^{n} \mathbb{E}_{\mathbf{v}} c_i$. For $i \in I$ (where I is still the set of i such that $\hat{T}_i$ is costly), let $d_i = (1-\hat{\sigma}) + \left(\hat{r}_{\text{space}} - \sigma_\varepsilon - \frac{f_i}{\hat{s}}\right)$ (i.e., to get d_i from c_i when $i \in I$, we replace $p_i(\mathbf{v})$ with f_i and subtract σ_ε); let $d_i = 0$ for $i \notin I$. Note that $\mathbb{E}_{\mathbf{v}} c_i \ge (1-\varepsilon)d_i$ (because $c_i \ge d_i$ as long as $p_i(\mathbf{v}) \le \sigma_\varepsilon \cdot \hat{s} + f_i$, which happens with probability $(1-\varepsilon)$ by the same reasoning as in Equation (2)). For every $i \notin I$, $f_i > \hat{s}(\hat{r}_{\text{space}} - \sigma_\varepsilon)$ by definition of I. Since $\sum_{i=1}^{c} f_i \le r_{\text{space}}s = r_{\text{space}}c\hat{s}$ (by Lemma 4.4),

$$\sum_{i\in I} f_i = \sum_{i=1}^{c} f_i - \sum_{i\notin I} f_i < \hat{s}(r_{\text{space}}c - (c-|I|)(\hat{r}_{\text{space}} - \sigma_\varepsilon))\,.$$

This gives

$$\begin{aligned}
\sum_{i=1}^{c} \mathop{\mathbb{E}}_{\mathbf{v}} c_i &\ge \sum_{i\in I} d_i \\
&= |I|(1-\hat{\sigma}+\hat{r}_{\text{space}}-\sigma_\varepsilon) - \sum_{i\in I}\frac{f_i}{\hat{s}} \\
&> |I|(1-\hat{\sigma}+\hat{r}_{\text{space}}-\sigma_\varepsilon) - r_{\text{space}}c + (c-|I|)(\hat{r}_{\text{space}}-\sigma_\varepsilon) \\
&= |I|(1-\hat{\sigma}) + c(\hat{r}_{\text{space}} - \sigma_\varepsilon - r_{\text{space}}) \\
&\ge c\left(\left(1 - \frac{r_{\text{space}}}{\hat{r}_{\text{space}} - \varepsilon}\right)\cdot(1-\hat{\sigma}) + (\hat{r}_{\text{space}} - \sigma_\varepsilon - r_{\text{space}})\right)
\end{aligned}$$

(by Equation (5)) and thus concludes the proof of Theorem 4.5. □

We now plug in parameters for a concrete graph.

Corollary 4.6. *There is a PoS T with output of size $s \approx 1.5 \cdot 10^9 \approx 2^{30.5}$ nodes that, in the pure pebbling model, has $\rho_{\text{hardness}} > 0.15$ for $r_{\text{space}} = 0.6$ and $q = 1$ such that*

- *For $r_{\text{space}} = 0.2$, the expected work of the pure pebbler is 15 time more than that the work of the entangled pebbler, while the storage is the same*
- *For $r_{\text{space}} = 0.6$, the expected work of the pure pebbler is 5 times more than that the work of the entangled pebbler, while the storage is 3 times more.*

[2]This can be seen by the following general argument. First, place $p_i(\mathbf{v})$ pebbles on $\hat{T}$ and find a set of $(1-\hat{\sigma})\hat{s}$ nodes of $\hat{Y}_i$ that trigger $\hat{t}_{\text{init}}\hat{r}_{\text{hardness}}$ work (which exist by the soundness of $\hat{T}_i$). Now repeat the following until the total number of pebbles reaches $\hat{r}_{\text{space}}\hat{s}$: place a new pebble on one of the nodes that triggers work; by the soundness of the $\hat{T}_i$, another node that triggers work must be found in its stead. This will eventually give $\hat{r}_{\text{space}}\hat{s} - p_i(\mathbf{v})$ additional nodes that trigger work.

For comparison, this PoS has about 50% more output nodes and costs about 4 times more to initialize than that the SDR graph deployed in Filecoin [GN23; Pro23].

Not that the performance of T (namely, $r_{\text{space}} = 0.6$ with $\rho_{\text{hardness}} = 0.15$) is similar to the deployed SDR graph, which has $r_{\text{space}} = 0.8$ and $r_{\text{hardness}} \approx 0.2$ but high soundness error $\sigma = 0.9$ ([GN23; Rey23]).

Proof. We plug in the following parameters to Theorem 4.5. Set $k = 32$ and $d = 5$ and thus $c = 33 \cdot 4 = 132$. Set $\hat{T}$ be the PoS from [Rey23, Appendix C] with $\ell = 55$ layers and each layer of size $\hat{s} = |\hat{Y}| = \binom{131}{4} \approx 1.1 \times 10^6 \approx 2^{23.5}$. Thus, each $\hat{T}$ is a graph of $\hat{t}_{\text{init}} = \ell\hat{s} = 55 \cdot \binom{131}{4} \approx 6.4 \times 10^8 \approx 2^{29.3}$ nodes. By [Rey23, Theorem 5], for $\hat{r}_{\text{space}} = 0.8$ and $\hat{q} = 1$, we have $\hat{\sigma} \leq 0.9$ and $\hat{r}_{\text{hardness}} \geq 0.784$. Multiplying by 132 and adding 7 layers of butterfly graphs gives us $\approx 9.6 \cdot 10^{10} \approx 2^{36.4}$ total nodes in T, with $Y \approx 1.5 \cdot 10^9 \approx 2^{30.5}$ output nodes. The total graph size $|T|$, which is important for initialization, is $62s$ instead of $11s$ in the deployed SDR; but because the initialization cost is proportional to the number of edges, and all but the bottom 8 layers of $\hat{T}$ have low degree (only the bottom 8 layers of $\hat{T}$ need horizontal edges for predecessor robustness [Rey23, Appendix C]; the remaining layers, except the top one, need only inter-layer edges), the initialization cost is only about 4 times more per output node.

We obtain that the entangled pebbler, for $q = 1$ and $r_{\text{space}} = \frac{1}{5}$, will have $\rho_{\text{hardness}} = \frac{1}{33}$ by Lemma 4.2. For the same parameters, the pure pebbler will have $\rho_{\text{hardness}} \geq .455 > \frac{15}{33}$ (per equation (8) using $r_{\text{space}} = 0.2$ and $\sigma_\varepsilon = 10^{-6}$), and will thus have to do at least 15 times more work than the entangled pebbler. In fact, even if the pure pebbler were to use triple the storage of the entangled pebbler (i.e., $r_{\text{space}} = \frac{3}{5}$), it would still need to perform 5 times as much work as the entangled pebbler. □

5 The Pebbling vs. Random Oracle Separation for Memory Hard Functions

In this section, we prove that Construction 3.8 gives a memory-hard function which yields a separation between a pure pebbling strategy and an entangled one.

Theorem 5.1. *Construction 3.8 provides a family of DAG-based dMHFs $\{F_k\}_{k\in\mathbb{N}}$ (with DAGs of indegree 2), such that:*

- $\text{CPC}(F_k) = \Omega\left(\frac{n^2}{\log n}\right)$,
- *but* $\text{CPC}^{\updownarrow}(F_k) = O\left(\frac{n^2 \log\log n}{\log^2 n}\right)$.

where n is the size of the dMHF.

Note that this separation in CPC immediately yields the same separation in CMC. That is, there exist memory hard functions for which storing XORs of labels asymptotically saves a factor $\Omega\left(\frac{\log n}{\log\log n}\right)$ of CMC over any algorithm that treats random oracle outputs atomically.

For easy reference, we recall the relevant parameters and how they relate to each other. In Construction 3.8 we let $d = k$, $c = (k+1)(d-1)$, and $\hat{s} = \binom{c-1}{d-1}$. We use $\hat{s}'$ to refer to $\hat{s}$ rounded up to the next power of 2. The graph G consists of c copies of $\hat{T}$ of size $\Theta(\hat{s}\log\hat{s})$ each along with

$\lceil \log k(d-1) \rceil$ layers of query connector nodes. We use s to denote the size of the final layer of G; there are $s\lceil \log k(d-1) \rceil$ nodes in all the query connectors (not counting the query connector sources). Note that $c\hat{s} \leq s < 2c\hat{s}$ (the exact value depends on how close $k(d-1)$ is to next power of 2), and therefore $\log s = \Theta(\log \hat{s})$ (because $c < \hat{s}$).

The full construction consists consists of a static graph G followed by one data-dependent line of length $s\delta = s \log \hat{s}'$. This makes the total size $n = \Theta(c\hat{s} \log \hat{s} + s\lceil \log k(d-1) \rceil + s \log \hat{s}) = \Theta(s \log \hat{s})$. Expressing the parameters in terms of the size n yields

$$\log \hat{s} = \Theta(\log n)\,,\; s = \Theta(c\hat{s}') = \Theta(c\hat{s}) = \Theta\left(\frac{n}{\log n}\right)\,,\; \log n > k = d = \Omega\left(\frac{\log n}{\log \log n}\right)\,.$$

We start by arguing that there is an efficient strategy for the entangled pebbler.

Lemma 5.2. *An entangled pebbler can pebble the sink of Construction 3.8 in cumulative space* $O\left(\frac{n^2 \log \log n}{\log^2 n}\right)$.

Proof. We wish to simulate the entangled pebbling strategy from Lemma 4.2. We start by pebbling all the sinks of G, which by Observation 3.9 can be done in cumulative space $O(n)$. We then store $c\hat{s}/d = O\left(\frac{n \log \log n}{\log^2 n}\right)$ entangled pebbles according to Strategy 3.5 and forget everything else.

We can now start pebbling the line L (i.e., answering the random queries of the pebbling game), maintaining our size $c\hat{s}/d$ configuration. Every $\delta = \log \hat{s}'$ steps we need to answer a random challenge, which can be done by recomputing $d-1$ copies of $\hat{T}$ as well as the query connector whose sink was queried. Recall that Construction 3.6 can be pebbled in CPC at most $2\hat{s}' \log \hat{s}'$, meaning $d-1$ copies can be pebbled in CPC at most $2(d-1)\hat{s}' \log \hat{s}'$. The query connectors are butterfly graphs of depth $\lceil \log k(d-1) \rceil$, meaning that, starting with all sources pebbled, a challenged sink can be pebbled in CPC at most $2(d-1)k$ by pebbling the binary tree leading to the sink. Thus the total CPC required to answer a challenge is at most

$$\begin{aligned} 2(d-1)\hat{s}' \log \hat{s}' + 2(d-1)k &= \frac{2c\hat{s}'}{k+1} \log \hat{s}' + O(\log^2 n) \\ &\leq \frac{4s}{k+1} \log \hat{s}' + O(\log^2 n) \\ &= O\left(\frac{n \log \log n}{\log n}\right)\,. \end{aligned} \tag{9}$$

We also need to take into account the CPC of maintaining the $c\hat{s}/d$ entangled pebbles during this computation. The parallel time it takes to pebble one length $\log \hat{s}$ segment of the line and answer the challenge is the length of the segment, i.e., $\log \hat{s}'$, plus the depth of $\hat{T}$, i.e., $2 \log \hat{s}'$, plus the depth of the query connector, i.e., $\log \lceil k(d-1) \rceil$. The total (parallel) time to pebble one segment is thus at most

$$\log \hat{s}' + 2 \log \hat{s}' + \log \lceil k(d-1) \rceil = O(\log n)\,. \tag{10}$$

Maintaining our size $c\hat{s}/d$ configuration for one segment uses, therefore, cumulative space $\frac{c\hat{s}}{d} \cdot O(\log n) = O(\frac{n \log \log n}{\log n})$, the same as the cumulative space used to answer a single challenge (see Equation (9))). Since we have $s = O(n/\log n)$ such segments, the total cumulative space used is $O\left(\frac{n^2 \log \log n}{\log^2 n}\right)$. □

It remains to prove the lower bound in Theorem 5.1. We argue that each segment of the line graph from Construction 3.8 requires large CPC.

Lemma 5.3. *A pure pebbler, starting with any collection of pebbles stored, will incur* CPC *of at least* $\frac{s \log \hat{s}}{6}$ *to wait for* δ *steps and then answer a single query of the randomized pebbling game from Construction 3.8.*

Proof. During the delay δ, before the query is known, the pure pebbler can add and remove pebbles on the graph; let $p_{\min}$ represent the smallest number of pebbles on the graph during this time.

As in the proof of Theorem 4.3, we split the randomness of the challenge into two parts. We first sample a vector $\mathbf{v}$ selecting one sink in each of the query connectors, and then pick a random query connector $\mathbf{j} \sim [\binom{c}{d-1}]$ and query the chosen sink of the chosen query connector. Using the notation introduced before and after Lemma 4.4, let f_i be the total weight of actual and fractional pebbles in $\hat{T}_i$ and $p_i(\mathbf{v})$ be the number of virtual pebbles in $\hat{T}_i$ after $\mathbf{v}$ is fixed. Note that $\mathbb{E}_{\mathbf{v}}[p_i(\mathbf{v})] \leq f_i$ for every i, where we get inequality rather than the equality of Lemma 4.4 since the query connectors are source-truncated. Let $I(\mathbf{j}) \subseteq [c]$ be the (indices of) the set of $\hat{T}_i$s queried by $\mathbf{j}$.

Once $\mathbf{v}$ is fixed, for every i, we can apply Claim 3.7 to the virtual pebble configuration on $\hat{T}_i$ to lowerbound the pure pebbler's expected cost of pebbling a random node in $\hat{Y}_i$. Using the fact that $I(\mathbf{j})$ and $\mathbf{v}$ are independent, we find that the CPC of a random challenge, measured from the last timestep where only $p_{\min}$ pebbles were in the configuration, is at least

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathbf{j},\mathbf{v}}\left[\sum_{i \in I(\mathbf{j})} (\hat{s} - p_i(\mathbf{v})) \log \hat{s}\right] &= \mathop{\mathbb{E}}_{\mathbf{j},\mathbf{v}}\left[\sum_{i \in [c]} \mathbb{1}_{i \in I(\mathbf{j})} (\hat{s} - p_i(\mathbf{v})) \log \hat{s}\right] \\
&= \sum_{i \in [c]} \mathop{\mathbb{E}}_{\mathbf{j}}[\mathbb{1}_{i \in I(\mathbf{j})}] \cdot \mathop{\mathbb{E}}_{\mathbf{v}}[(\hat{s} - p_i(\mathbf{v})) \log \hat{s}] \\
&\geq \sum_{i \in [c]} \frac{k}{k+1} (\hat{s} - f_i) \log \hat{s}. \\
&> \frac{k}{k+1} \left(\frac{s}{2} - p_{\min}\right) \log \hat{s} \\
&\geq \frac{1}{4}(s - 2p_{\min}) \log \hat{s}\,,
\end{aligned}$$

where the second to last step follows from the fact that $s < 2c\hat{s}$.

Moreover, the pure pebbler also has at least $p_{\min}$ in memory for $\delta = \log \hat{s}'$ timesteps, which requires a CPC of at least $p_{\min} \log \hat{s}$. We can thus bound the CPC of any pure pebbler to be at least

$$\max\left(\frac{1}{4}(s - 2p_{\min}) \log \hat{s},\, p_{\min} \log \hat{s}\right) \geq \frac{s \log \hat{s}}{6}\,.$$

This concludes the proof of Lemma 5.3. □

Since we have s segments in L (i.e., play the randomized pebbling game for s iterations), a pure pebbler needs CPC at least

$$\frac{s^2 \log \hat{s}}{6} = \Omega\left(\frac{n^2}{\log n}\right)$$

to compute the function constructed in Construction 3.8, proving Theorem 5.1.

Acknowledgements

David Engström and Susanna F. de Rezende received funding from the Knut and Alice Wallenberg grant KAW 2023.0116, ELLIIT, and the Swedish Research Council grant 2021-05104. They also gratefully acknowledge that they have benefited greatly from being part of Basic Algorithms Research Copenhagen (BARC) environment financed by the Villum Investigator grant 54451.

References

[ABFG14] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. "Proofs of Space: When Space Is of the Essence". In: *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*. Ed. by Michel Abdalla and Roberto De Prisco. Vol. 8642. Lecture Notes in Computer Science. Springer, 2014, pp. 538–557. URL: https://doi.org/10.1007/978-3-319-10879-7%5C_31.

[ACKKPT16a] Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. *On the Complexity of Scrypt and Proofs of Space in the Parallel Random Oracle Model*. Cryptology ePrint Archive, Paper 2016/100. 2016. URL: https://eprint.iacr.org/2016/100.

[ACKKPT16b] Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. "On the Complexity of Scrypt and Proofs of Space in the Parallel Random Oracle Model". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 358–387. URL: https://doi.org/10.1007/978-3-662-49896-5%5C_13.

[ACPRT17] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. "Scrypt Is Maximally Memory-Hard". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10212. Lecture Notes in Computer Science. 2017, pp. 33–62. URL: https://doi.org/10.1007/978-3-319-56617-7%5C_2.

[AS15] Joël Alwen and Vladimir Serbinenko. "High Parallel Complexity Graphs and Memory-Hard Functions". In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. Ed. by Rocco A. Servedio and Ronitt Rubinfeld. ACM, 2015, pp. 595–603. URL: https://doi.org/10.1145/2746539.2746622.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. "Proofs of Space". In: *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. Ed. by Rosario Gennaro and Matthew Robshaw. Vol. 9216.

Lecture Notes in Computer Science. Springer, 2015, pp. 585–605. URL: https://doi.org/10.1007/978-3-662-48000-7%5C_29.

[DKW11a] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. "Key-Evolution Schemes Resilient to Space-Bounded Leakage". In: *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Springer, 2011, pp. 335–353. URL: https://doi.org/10.1007/978-3-642-22792-9%5C_19.

[DKW11b] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. "One-Time Computable Self-erasing Functions". In: *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*. Ed. by Yuval Ishai. Vol. 6597. Lecture Notes in Computer Science. Springer, 2011, pp. 125–143. URL: https://doi.org/10.1007/978-3-642-19571-6%5C_9.

[DNW05] Cynthia Dwork, Moni Naor, and Hoeteck Wee. "Pebbling and Proofs of Work". In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, 2005, pp. 37–54. URL: https://doi.org/10.1007/11535218%5C_3.

[Fis19] Ben Fisch. "Tight Proofs of Space and Replication". In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Lecture Notes in Computer Science. Springer, 2019, pp. 324–348. URL: https://doi.org/10.1007/978-3-030-17656-3%5C_12.

[GN23] Irene Giacomelli and Luca Nizzardo. *Filecoin Proof of Useful Space — Technical Report*. https://drive.google.com/file/d/1notObdkPT1BCztgspIpzSUAzWSrM8h81/view. 2023.

[KK14] Nikolaos P. Karvelas and Aggelos Kiayias. "Efficient Proofs of Secure Erasure". In: *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*. Ed. by Michel Abdalla and Roberto De Prisco. Vol. 8642. Lecture Notes in Computer Science. Springer, 2014, pp. 520–537. URL: https://doi.org/10.1007/978-3-319-10879-7%5C_30.

[Per09] C. Percival. "Stronger key derivation via sequential memory-hard functions". In: *BSDCan 2009*. 2009.

[Pie19] Krzysztof Pietrzak. "Proofs of Catalytic Space". In: *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, San Diego, California, USA, January 10-12, 2019*. Ed. by Avrim Blum. Vol. 124. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 59:1–59:25. URL: https://doi.org/10.4230/LIPIcs.ITCS.2019.59.

[Pro23] Protocol Labs. *Filecoin Spec. Algorithms. Stacked DRG PoRep*. https://spec.filecoin.io/algorithms/sdr/. 2023.

[Rey23] Leonid Reyzin. *Proofs of Space with Maximal Hardness*. Cryptology ePrint Archive, Paper 2023/1530. 2023. URL: https://eprint.iacr.org/2023/1530.

[Rey24] Leonid Reyzin. "Proofs of Space with Maximal Hardness". In: *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*. IEEE, 2024, pp. 1159–1177. URL: https://doi.org/10.1109/FOCS61266.2024.00076.