Provably Memory-Hard Proofs of Work With Memory-Easy Verification

Jeremiah Blocki[0000−0002−5542−4674] and Nathan Smearsoll[0000−0002−5957−6616]

Purdue University, West Lafayette IN 47907, USA

Abstract. A Proof of Work (PoW) is an important construction for spam-mitigation and distributed consensus protocols. Intuitively, a PoW is a short proof that is easy for the verifier to check but moderately expensive for a prover to generate. However, existing proofs of work are not egalitarian in the sense that the amortized cost to generate a PoW proof using customized hardware is often several orders of magnitude lower than the cost for an honest party to generate a proof on a personal computer. Because Memory-Hard Functions (MHFs) appear to be egalitarian, there have been multiple attempts to construct Memory-Hard Proofs of Work (MHPoW) which require memory-hard computation to generate, but are efficient to verify. Biryukov and Khovratovich (Usenix, 2016) developed a MHPoW candidate called Merkkle Tree Proofs using used the Argon2d MHF. However, they did not provide a formal security proof and Dinur and Nadler (Crypto, 2017) found an attack which exploited the data-dependencies of the underlying Argon2d graph.
We revisit the security of the MTP framework and formally prove, in the parallel random oracle model, that the MTP framework is sound when instantiated with a suitable *data-independent* Memory-Hard function. We generically lower bound the cumulative memory cost (cmc) of any prover for the protocol by the pebbling cost of the *ex-post facto* graph. We also prove that as long as the underlying graph of the original iMHF is sufficiently depth-robust that, except with negligible probability, the *ex-post facto* will have high cumulative memory cost (cmc). In particular, if we instantiate the iMHF with DRSample then we obtain a MHPoW with the following properties: (1) An honest prover for the protocol can run in sequential time $O(N)$, (2) The proofs have size $\texttt{polylog}(N)$ and can be verified in time $\texttt{polylog}(N)$ (3) Any malicious prover who produces a valid proof must incur high cumulative memory complexity at least $\Omega\left(\frac{N^2}{\log N}\right)$. We also develop general pebbling attacks to which we use to show that (1) any iMHF based MHPoW using the MTP framework has proof size at least $\Omega\left(\log^2 N/\log\log N\right)$, and (2) at least $\tilde{\Omega}(N^{0.32})$ when the iMHF is instantiated with Argon2i, the data-independent version of Argon2.

1 Introduction

Proofs of Work (PoWs), originally introduced by [DN93] to combat spam email, allow a prover to convince a verifier that they have spent significant computational work in evaluating a function upon some string, such as an email message

or list of transactions on a blockchain. In the spam reduction application, before sending an e-mail χ the sender would generate a proof of work $\pi_\chi = \texttt{Prove}(\chi)$ for the string χ and send both χ and π_χ to the recipient. The recipient would discard the email χ unless the proof of work for is valid i.e., $\texttt{Verify}(\chi, \pi_\chi) = 1$. Importantly, it should be moderately expensive to generate a proof π_χ for a string χ, but the final proof π_χ should be short and inexpensive to verify. PoWs can also be used to protect against distributed denial of service attacks and in many blockchain based distributed consensus protocols.

Hashcash [B+02] is one of the simplest and most natural PoW schemes and the backbone of the Bitcoin consensus mechanism. Fixing a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ and a hardness parameter κ the Hashcash PoW algorithm $\texttt{Prove}^{\mathcal{H}}(\chi, \kappa)$ simply searches for a string π_χ such that $\mathcal{H}(\chi, \pi_\chi)$ has κ leading 0's i.e., $\mathcal{H}(\chi, \pi_\chi) = x \circ 0^\kappa$ for some string $x \in \{0,1\}^{\lambda-\kappa}$. The parameter κ controls the hardness of the PoW i.e., in the random oracle model any (possibly malicious) proving algorithm will need to make 2^κ queries in expectation to generate a valid proof for χ. The Hashcash verification algorithm is incredibly efficient. Given a proof π_χ the verification algorithm $\texttt{Verify}^{\mathcal{H}}(\chi, \pi_\chi, \kappa)$ simply computes $y = H(\chi, \pi_\chi)$ checks that y has κ leading 0's.

Unfortunately, the Hashcash PoW is not egalitarian — meaning that the cost of generating a proof of work on customized hardware can be orders of magnitude cheaper than the cost of generating a proof on a personal computer. For example, Application-Specific Integrated Circuits (ASICs) for computing the SHA-256 hashing algorithm are four to five orders of magnitude more energy efficient than the CPU on a state of the art personal computer e.g., see [RD17,NBF+16]. This is highly undesirable for applications like spam mitigation as there is no good way to tune the hardness parameter κ. Either it becomes prohibitively expensive for legitimate senders to generate a proof of work on their personal computer, or it will be relatively cheap for a spamming attacker to generate proofs of work on a customized ASIC.

Biryukov and Khovratovich [BK16] proposed developing egalitarian proofs of work: proofs of work where the cost of generating a proof using customized hardware is not significantly better than on general-purpose hardware. Motivated by the observation that memory costs tend to be egalitarian across different architectures they introduced a candidate *memory-hard* proof of work (MHPoW) called Merkle Tree Proofs (MTP) which combined Merkle Trees with the Argon2 Memory-Hard Function [BDK16]. Intuitively, the prover first runs the Argon2 evaluation algorithm on input χ to generate N memory blocks $L_1, \ldots, L_N$ using a random oracle $\mathcal{H}$ where $N = 2^n$ denotes the running time parameter. Then the prover computes the root τ of a Merkle-Tree commitment to the labels $L_1, \ldots, L_N$, and applies the Fiat-Shamir paradigm to generate k challenges $c_1, \ldots, c_k$ by hashing the root τ e.g., $c_i = 1 + (\mathcal{H}(i, \tau) \mod N)$. For each challenge c_i uses the Merkle-Tree to prove that the label c_i is locally consistent by revealing L_{c_i} and any dependent label value L_u used which was used to compute L_{c_i}. The final proof π_χ consists of the Merkle-Tree root τ along with the Merkle-Tree openings for each challenge label L_{c_i} and depended labels.

The MTP construction [BK16] seems to be natural as the Argon2 MHF is believed to have maximal cumulative memory complexity $\Omega(N^2)$, and the challenges ensure that almost all labels are computed in the expected way. However, [BK16] did not prove that their MTP construction satisfied any formal notion of memory-hardness. Dinur and Nadler [DN17] found an attack on MTP which significantly reduces the cost of generating a proof e.g., by a factor of $55,000$. Biryukov and Khovratovich [BK15] proposed a different MHPoW candidate called Equihash along with an argument that the construction was secure. However, Alcock and Ren [AR17] demonstrated a flaw in the security analysis of Equihash raising questions about whether or not Equihash is truly memory-hard/egalitarian.

This leads us to our central question:

Can a memory-hard proof of work scheme be designed which is provably secure?

One naive solution to the designing egalitarian memory-hard proofs of work would be to simply have the prover compute a memory hard function f with the input χ and send the output $\pi = f(\chi)$ to the verifier. We have constructions of memory hard functions which are *provably* memory-hard (i.e., have high cumulative memory complexity) such as SCRYPT [ACP+17] and DRSample [ABH17] in the parallel random oracle model. While the solution ensures that any prover that generates a valid MHPoW would need to perform memory-hard computation, the problem is that the verifier would also need to perform expensive memory-hard computation.

An alternate solution would be to have the prover compute $y = f(\chi)$ and then generate a succinct non-interactive argument of knowledge (SNARK[BCG+13]) π_χ proving that the output is correct. However, the overhead would likely be quite high in practice. Furthermore, it is not clear that the solution would still be egalitarian. In particular, the cost for an honest party to generate the SNARK proof π_χ is substantially higher than the cost to evaluate the MHF $f(\chi)$ i.e., prover costs would dominated by the cost to generate the SNARK. The cumulative memory cost to generate the SNARK proof π_χ would likely be substantially lower for a parallel attacker using customized hardware making the overall solution non-egalitarian.

Another alternative solution would be to use trapdoor-memory hard functions such as td-SCRYPT [BP17]. Auerbach et al. [AGP24] proved that, without the trapdoor, the cumulative memory complexity was at least $\Omega(N^2/\log N)$ and Blocki and Lee [BL24] improved the lower bound to $\Omega(N^2)$. Any party who has the trapdoor can evaluate the function at substantially reduced cost $\tilde{O}(N)$. Thus, a verifier who has the trapdoor could validate the claim that $y = f(\chi)$ at greatly reduced cost. However, the solution no long qualifies as a Proof of Work because there is no efficient public verifier i.e., verifiers that do not have the trapdoor will need perform memory hard computation to check that $y = f(\chi)$. If the prover wants a proof that can be efficiently checked by two different verifiers s/he would need to generate two separate proofs. Furthermore, even for a verifier that has the trapdoor the solution is not as efficient as we would like. Ideally we would

like the MHPoW verification algorithm to run in time $O(\texttt{polylog}(N))$. By contrast, computing the td-SCRYPT function with the trapdoor requires at least $\Omega(N)$ steps.

Scientific understanding of Memory-Hard Functions has advanced significantly over the past decade, both in formal mathematical definitions to quantify "memory-hardness" [AS15,ABP18,RD17], techniques to analyze memory-hard functions [AB16,AB17,ABP17,BZ17,AGK+18,BHK+19] and techniques to construct memory-hard functions. Alwen and Serbinekno [AS15] pointed out that the classical notion of space-time complexity is not suitable for defining memory-hardness [ABP17,ABH17,BHK+19]. In particular, a function with high space-time complexity does not necessarily have high amortized space-time complexity i.e., a parallel attacker may be able to evaluate the function on multiple inputs without any increase to its space-time costs. Thus, Alwen and Serbinekno [AS15] introduced the notion of cumulative memory complexity to quantify the amortized space-time complexity of a memory-hard function.

While the definition of cumulative memory complexity (or equivalently amortized space-time complexity), extends naturally to our the Memory-Hard Proof of Work setting we still lack techniques to analyze the security of MHPoW candidates. Alwen and Serbinenko also proved that in the parallel random oracle model the cumulative memory complexity of a data-independent memory hard function (iMHF) $f_{G,\mathcal{H}}$, a class memory-hard which provides natural resistance to side-channel attacks, is fully captured by the cumulative pebbling cost of the underlying data-dependency graph G. This powerful connection between cumulative memory complexity and graph pebbling greatly simplifies the task of cryptanalyzing iMHF candidates. has been exploited to develop attacks on iMHF candidates [AB16,AB17,ABP17,BZ17,AGK+18,BHK+19], and to develop iMHFs[ABP17,ABH17,BHK+19] with provably high cumulative memory complexity by constructing depth-robust graphs with provably high cumulative pebbling cost. However, it is unclear whether or not one could establish a similar connection between graph pebbling and the cumulative memory complexity of a MHPoW. This leads us to our second question:

Can we relate the cumulative memory complexity of a MHPoW to the pebbling cost of its DAG?

1.1 Our Contributions

We prove that the Merkle-Tree Proof framework yields a secure Memory-Hard Proof of Work when instantiated with a suitable data-independent memory hard function. A data-independent memory-hard function $f_{G,H}$ can be specified using a random oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ and a directed acyclic graph G with N nodes and constant indegree. Applying the Merkle-Tree Proof framework to $f_{G,H}$ we obtain a MHPoW $(\texttt{Prove}^H, \texttt{Verify}^H)$ such that (1) the honest prover algorithm $\texttt{Prove}^H(\chi, N)$ has $\texttt{cmc}$ at most $O(N^2\lambda)$ and generates a proof of size $O(k\lambda \log N)$, (2) the honest verifier makes $O(k \log N)$ queries to the hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$ and (3) except with negligible probability any

dishonest prover $\mathtt{Prove}'$ in the parallel random oracle model must incur cumulative memory cost at least $\mathsf{cmc}(\mathtt{Prove}') = \Omega(\mathsf{cc}(G-S)\lambda)$ where $|S| \leq N\lambda/k$. Here, $G-S$ denotes the graph G after removing all nodes in $S \subseteq V(G)$ and any incident edges. The notation $\mathsf{cc}(G-S)$ denotes the cumulative pebbling cost of the best parallel pebbling of $G-S$.

For an iMHF $f_{G,H}$ Alwen and Serbinenko [AS15] proved that $\mathsf{cmc}(f_{G,H}) = \Omega(\mathsf{cc}(G)\lambda)$ i.e., any evaluation algorithm for $f_{G,H}$ has cumulative memory cost lower bound directly by the cumulative pebbling cost of G. In our setting a cheating prover may never actually compute $f_{G,H}$ i.e., because the verifier cannot double check all of the prover's computations a cheating prover may make up the values for some intermediate labels without getting caught. However, we can conclude that $\mathsf{cmc}(\mathtt{Prove}') \geq \Omega\left(\min_{S:|S|\leq N\lambda/k} \mathsf{cc}(G-S)\lambda\right)$. Thus, a suitable iMHF DAG G should have the property that $\mathsf{cc}(G-S)$ is large for all reasonably large sets $S \subseteq V(G)$. We show that this property hold as long as the graph G is instantiated by a depth-robust graph such as DRSample [ABH17]. In particular, the DRSample graph has the property that, for some suitable constant $c > 0$ and any set S of at most $|S| \leq \frac{cN}{\log N}$ nodes, the subgraph $G' = G - S$ still has cumulative complexity $\Omega\left(\frac{N^2}{\log N}\right)$, which is asymptotically equivalent to the original DRSample graph G [ABH17,BHK+19]. Thus, setting $k = O(\lambda \log N)$ we obtain a MHPoW with proof size $O(\lambda^2 \log^2 N)$ such that the verifier only makes $O(\lambda \log^2 N)$ queries to the random oracle H. Any cheating prover $\mathtt{Prove}'$ must incur cmc at least $\Omega\left(\frac{N^2\lambda}{\log N}\right)$.

We also show that instantiating the MTP MHPoW framework with the Argon2i[BDK16] graph G yields provable guarantees. However, the proof size must be larger, and the cmc guarantee will be much weaker e.g., if we are willing to tolerate a proof size of $\Omega\left(N^{1/3}\lambda\right)$ then we can ensure that any cheating prover incurs cmc at least $\widetilde{\Omega}\left(N^{5/3}\lambda\right)$.

Finally, we analyze the inherent limitations of MTP-style constructions by developing pebbling-based attacks on MTP MHPoWs. The attacker first picks a set S of nodes to "delete" from the graph and then finds a low cost pebbling for $G-S$. As long as the set $|S| \leq N/k$ is not too big we can construct a cheating prover $\mathtt{Prove}'$ which succeeds with constant probability and has cumulative memory complexity at most $\mathsf{cmc} \leq \mathsf{cc}(G-S)\lambda$. Thus, by finding pebbling attacks we can lower bound the parameter k (number of challenges) and the size of the proof necessary to prevent attacks with lower than desired cumulative memory complexity. Specific findings include the following:

- In general for any DAG G with N nodes and constant indegree if $k = o(\log N/\log\log N)$ then there is a set $|S| \leq N/k$ such that $\mathsf{cc}(G-S) = o(N^2/\log^c N)$ for any constant $c \geq 1$. In particular, for any graph G that we use to instantiate our MTP-style MHPoW our proof size must be at least $\Omega(\lambda \log^2 N/\log\log N)$ or else there will be a cheating prover with lower cumulative memory complexity than we hope e.g., $\mathsf{cmc} = o(N^2\lambda/\log^{1000} N)$.
- For the Argon2i DAG [BDK16] we show that for any $k \geq 1$ there is a set $|S| \leq N/k$ such that $\mathsf{cc}(G-S) = O(Nk^{12/5})$. If $k = o\left(N^{5/18}\right)$ then

$\mathsf{cc}(G - S) = o(N^{5/3})$ and the cc bound approaches N as k gr smaller. Thus, when instantiating our MTP MHPoW with Argon2i we must pick an undesirably large k (leading to larger proof sizes) just to ensure the the cmc is not $O(\lambda N \mathsf{polylog} N)$.

1.2 Related Work

Merkle Trees and Their Applications to Proofs of Space and Proofs of Sequential Work A Merkle Tree, introduced by [Mer88], is a short commitment to a large set of strings. Given a particular commitment, one can later reveal any of the strings used, to show that their value was indeed committed to. Merkle Trees have been used previously to construct Proofs of Space (PoS) [DFKP15] and Proofs of Sequential Work (PoSW)[MMV13]. Constructions of PoS [DFKP15] and PoSWs [MMV13] are similar to our construction of a Memory-Hard Proof of Work. In particular, constructions of PoSW's [MMV13,CP18] and PoS [DFKP15,Pie19] involve using a directed acyclic graph and a cryptographic hash function (Random Oracle) to generate labels for each node in the graph. Similarly to our construction, the prover commits to the labels using a Merkle-Tree structure then proves to the verifier that most nodes in the graph are *locally consistent* i.e., when challenged to prove that a node v is locally consistent the prover will use the Merkle-Tree to reveal the label of node v as well as the label of v's parents. In both cases these consistency challenges can be generated non-interactively by invoking the Fiat-Shamir paradigm — Proofs of Space require a separate interactive audit phase where the verifier periodically challenges the prover to confirm that the prover has still allocated space on its hard disk. One can formally prove that PoSW constructions[MMV13,CP18] are secure in the parallel random oracle model — these proofs were later extended to the parallel quantum random oracle model[BLZ21,CFHL21]. While the notion of PoSW security is quite different from Memory-Hard Proofs of Work some of analysis techniques are useful in our context e.g., the notion of a "bad-order" event in the analysis of [CP18] and the partial Merkle-Tree extraction technique from [BLZ21].

Merkle Tree Proofs Biryukov and Khovratovich [BK16] proposed the first candidate construction of a Memory-Hard Proof of Work called Merkle Tree Proofs (MTP) following a similar approach to prior constructions of Proofs of Sequential Work (PoSW)[MMV13] and Proofs of Space [DFKP15,Pie19]. One important distinction for their construction was that the prover used their data-dependent memory-hard function Argon2d to generate the initial labels committed to by the Merkle-Tree. Because Argon2d is data-dependent the directed acyclic graph is not fixed a priori and the directed edges in the graph are dependent on intermediate labels. Biryukov and Khovratovich [BK16] did not provide any rigorous security proofs for their construction, and Dinur and Nadler [DN17] later gave an attack on MTP which exploited the fact that a malicious prover could modify the edge structure in a data-dependent graph by performing precomputation and then selectively modifying several of the intermediate labels to advantageously

control to edge structure. The attack of Dinur and Nadler [DN17] highlights the importance of rigorous security proofs for Memory-Hard Proofs of Work. We demonstrate the the Merkle Tree Proof framework used by Biryukov and Khovratovich [BK16] is sound as long as the underlying graph is fixed a priori (i.e., cannot be adversarially modified) and is sufficiently depth-robust.

Memory-Hard Functions The notion of a Memory-Hard Proof of Work is similar to that of a standard Memory-Hard Function (MHF). Alwen and Serbinenko [AS15] gave the first rigorous security definitions and proofs for a memory-hard function. In particular, they introduced the notion of *cumulative memory cost (CMC)* to model the *amortized space-time* complexity of a function. Their security analysis provided a more general pebbling reduction showing that the *cumulative memory cost* of data-independent Memory Hard Functions (iMHFs) was fully characterized by the pebbling cost of the underlying DAG. Alwen and Blocki [AB16] showed that any *any* directed-acyclic graph with N nodes and constant indegree had cumulative pebbling complexity at most $\Omega(N^2 \log\log N/\log N)$ and that known iMHF candidates such as Argon2i (winner of the password hashing competition[1]) had substantially lower cumulative pebbling complexity. On the positive side Alwen et al. [ABP17,ABH17] constructed depth-robust graphs with constant indegree and cumulative pebbling complexity *at least* $\Omega(N^2/\log N)$. These depth-robust graph are also useful for constructing Memory-Hard Proofs of Work.

1.3 Technical Overview

Our central technical contribution is to provide an extractor argument linking the cumulative memory complexity of a Memory Hard Proof of Work constructed using the Merkle Tree Proof framework to the pebbling complexity of the underlying directed acyclic graph G in the parallel random oracle model (PROM). Previously Alwen and Serbinenko provided a similar extractor argument linking the cumulative memory complexity of a data-independent Memory Hard Function (iMHF) to the cumulative pebbling complexity of the underlying DAG G in the PROM. We begin by reviewing the extractor argument of [AS15] highlighting additional challenges that arise in our Proof of Work context and how we overcome them.

Review: Parallel Pebbling, the Parallel Random Oracle Model, and Graph Labeling Given $G = (V = [N], E)$ with nodes $V = \{1, \ldots, N = 2^n\}$ given in topological order, a random oracle H, and an input string χ, we can label each vertex of G as follows: we label the first node $\ell_1 = H(\chi, 1)$, then label each node v with incomputing edges from $\{v_1, \ldots, v_k\} = \{u : (u, v) \in E\} \doteq \mathsf{parents}(v, G)$ by computing $\ell_v = H(\chi, v, \ell_{v_1}, \ldots, \ell_{v_k})$. A pebbling $P = (P_0, P_1, \ldots, P_t)$ of G is a sequence of sets $P_0, P_1, P_2, \ldots, P_t \subseteq P_t$ such that (1) $P_0 = \{\}$, (2) $\forall i < t, v \in P_{i+1} \setminus P_i$ we have $\mathsf{parents}(v, G) \doteq \{u : (u, v) \in E\} \subseteq P_i$ and (3)

[1] `https://www.password-hashing.net/`

$V \subseteq \bigcup_i P_i$. Intuitively, each node $v \in V$ in the graph represents an intermediate data-label ℓ_v generated during the computation of an iMHF $f_{G,H}$ and edges represent data-dependencies between these labels e.g., if $\ell_v = H(v, \ell_{v-1}, \ell_u)$ then the graph G would include directed edges $(v-1, v)$ and (u, v). Placing a pebbling on node v corresponds to computing label ℓ_v and storing it in memory, and pebbling constraint (2) corresponds to the natural requirement that labels ℓ_{v-1} and ℓ_u need to be available before computing ℓ_v. Constraint (1) implies that we begin computation without any of the labels in memory and constraint (3) requires that every label is computed eventually since we must eventually compute the label of the final sink node N. The cumulative pebbling complexity of a pebbling is $\mathtt{cc}(P) \doteq \sum_i |P_i|$ and the cumulative pebbling complexity of a graph G is $\mathtt{cc}(G) \doteq \min_P \mathtt{cc}(P)$ where the minimum is taken over all legal parallel pebblings of the graph G i.e., $\mathtt{cc}(G)$ is the cumulative pebbling cost of the best parallel pebbling of G.

Computation in the Parallel Random Oracle Model (PROM) takes place over rounds. A PROM algorithm $A(\sigma_0)$ begins execution in round $i = 0$ with an initial input state $\sigma_0 = (\tau_0, \{\})$. The algorithm then performs arbitrary computation before outputting a new state τ_1 and a batch of random oracles queries Q_1. The queries $Q_1 = \{x_{1,1}, \ldots, x_{1,z_1}\}$ are then submitted to the random oracle H to obtain responses $H(Q_1) = \{H(x_{1,1}), \ldots, H(x_{1,z_1})\}$. In round $i = 1$ we set $\sigma_1 = (\tau_1, H(Q_1))$ and $A(\sigma_1)$ may again perform arbitrary computation before outputting τ_2 and a new batch of ROM queries Q_2. More generally, in round i we run $A(\sigma_i = (\tau_i, H(Q_i))$ to generate our new state τ_{i+1} and batch of queries Q_{i+1}. Fixing the input χ, random oracle H and any random coins R used by our algorithm A we obtain a sequence of states $\mathtt{trace}(A, H, \chi, R) = \sigma_0, \sigma_1, \ldots, \sigma_t$ called the execution trace. The cumulative memory cost of an execution trace is $\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) = \sum_{i=1}^{t} |\sigma_i|$ where $|\sigma_i|$ denotes size of the state at time i (# bits).

Review of Alwen and Serbinenko Alwen and Serbinenko [AS15] showed that for any PROM algorithm A which computes $f_{G,H}(x)$ correctly with probability at least ϵ (the randomness is taken over the selection of the random oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ and any random coins R) that we will have $\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) \geq c\mathtt{cc}(G) \cdot \lambda$ with probability $\epsilon - \mu(\lambda)$ for some negligible function $\mu(\ldots)$ and some constant $c > 0$. To prove this Alwen and Serbinenko [AS15] first introduce the notion of an ex-post facto pebbling of G which can be extracted from the execution trace $\mathtt{trace}(A, H, \chi, R)$. In particular, for each node v they track rounds where the label ℓ_v appears as an input to a random oracle query and rounds where the "prelabel" $\mathtt{prelabel}(v, H) \doteq (\chi, v, \ell_{v_1}, \ldots, \ell_{v_\delta})$ is submitted as a query to the random oracle — the output of this query will be $H(\chi, v, \ell_{v_1}, \ldots, \ell_{v_\delta})$. A node v is said to be "needed" in round i if the label ℓ_v will be used as part of a random oracle input before the label ℓ_v will be generated as an output and we use the notation $\mathsf{need}(i) \subseteq V$ to denote the set of nodes that are needed during round i.

Alwen and Serbinenko define the ex-post facto pebbling by setting $P_i = \mathsf{need}(i)$ and they prove that, except with negligible probability, the ex-post facto

pebbling is legal[2]. To complete their proof Alwen and Serbinenko argue that, except with negligible probability, we will have $|\sigma_i| \geq c\lambda|P_i|$ for all rounds i i.e., the number $|P_i|$ of pebbles at round i is upper bounded by the size of the state $|\sigma_i|$.

To argue this, they give an extractor which, given a state σ_i and a short hint h, can extract $|P_i|$ query-output pairs without ever having submitted them as a query. Essentially, the extractor will simulate the PROM algorithm A with the initial state σ_i knowing that for every node $v \in P_i$ with parents $u_1, \ldots, u_\delta$ the corresponding label $\ell_v = H(\texttt{prelab}(v, H))$ will appear as an input to a future random oracle query made by A before its prelabel $\texttt{prelabel}(v, H)$. The hint h will encode the index of the first query where ℓ_v appears as an input as well as the index of the first subsequent query to $\texttt{prelabel}(v, H)$ (if applicable) — if A queries $\texttt{prelabel}(v, H)$ in the future the simulator will simply respond with with the extracted value ℓ_v without every querying the random oracle H. In this way the extractor can obtain $|P_i|$ input-output pairs $(\texttt{prelabel}(v, H), \ell_v = H(\texttt{prelabel}(v, H))$ for all $v \in P_i$ without ever querying the random oracle on the corresponding inputs $H(\texttt{prelabel}(v, H))$. This implies that the length of the input to the extractor must be $|\sigma_i| + |h| = \Omega(|P_i|\lambda)$. Otherwise, we would obtain a contradiction as it would be possible to compress the random oracle.

Challenge 1: Labels Are Not Fixed A Priori Our first challenge in proving the security of MTP is that the labeling may not be fixed a priori. Recall that a proof consists of a merkle tree root τ', the opening of k randomly selected leaf nodes $c_1, \ldots, c_k \in V$ in the merkle tree along with the opening of their parents. The verifier will check that all the Merkle Tree openings are valid and then verify that each challenge node c_i is "locally consistent" i.e., if ℓ'_{c_i} denotes the revealed value from the MT opening then we expect that $\ell'_{c_i} = H(v, \chi, \ell'_{v_1}, \ldots, \ell'_{v_\delta})$ where $v_1, \ldots, v_\delta$ are the parents of node v and ℓ'_{v_i} denotes the revealed value from the MT opening of leaf node v_i. Because the verifier only checks $k \ll N$ nodes are locally consistent a proof that is accepted does not necessarily have to use the correct labels $\ell_1, \ell_2, \ldots, \ell_N$. Thus, we cannot simply follow Alwen and Serbinenko [AS15] to define an ex-post facto pebbling by observing when each honest label ℓ_v appears as an input (or output) of a random oracle query.

We address this first challenge by introducing the notion of an ex-post facto labeling. In particular, once a PROM algorithm A outputs the root τ' of a Merkle-Tree we can extract a labeling $\ell'_1, \ldots, \ell'_n$ from τ' by examining the database D of all prior random oracle queries made by A. As long as D contains no collisions the ex-post facto labeling will be unique, and because $q \ll 2^{\lambda/2}$ the probability that D contains a collision will be negligible.

Challenge 2: Pebbling Graph is not Fixed A Priori Once we have such an ex-post facto labeling $\ell'_1, \ldots, \ell'_n$, it would be natural to define an ex-post facto pebbling

[2] The pebbling might be illegal if for some node v the label ℓ_v is used as a random oracle input before it appears as the output to a random oracle query. However, the probability of such an "out-of-order" event occurring is easily shown to be negligible.

from the execution trace of our PROM algorithm A by observing when label ℓ'_v appears as an input or as an output to a random oracle query. Unfortunately, the resulting pebbling sequence $P_1, \ldots, P_t$ may not be a legal pebbling sequence for the original graph G. The problem is that some nodes in the ex-post facto labeling may not be locally consistent e.g., for a node v with parents $v_1, \ldots, v_\delta$ we might have $\ell'_v = 0^\lambda$ instead of $\ell_{v'} = H(v, \chi, \ell'_{v_1}, \ldots, \ell'_{v_k})$. Because the verifier only checks $k \ll N$ random nodes for local consistency it is certainly possible that several nodes are not locally consistent. If v is not locally consistent then it is possible that the label ℓ'_v appears out of order e.g., ℓ'_v might appear as input to a random oracle query before appearing as an output. Thus, if the edge (v, w) exists it may be possible to compute $\ell_{w'}$ without ever producing ℓ'_v as a random oracle output — even if the node w was locally consistent.

We address this challenge by introducing the notion of an ex-post facto subgraph G' of G which can be extracted from the execution trace $\mathtt{trace}(A, H, \chi, R)$ of our PROM algorithm A. To define the ex-post facto subgraph G' we first introduce the notion of node colors. We call a node v *Green* if it is locally consistent and *Red* otherwise. We call the set of labels $\ell'_1, \ldots, \ell'_n$ actually used by the prover the Ex-Post-Facto Labeling (see definition 8), and can use them to check the color of each node.

Definition 1 (Informal, see Definition 7). *Given a directed acyclic graph G with nodes $V = [N]$ and a fixed labeling $\ell'_1, \ldots, \ell'_N \in \{0,1\}^\lambda \cup \{\perp\}$ (which may or may not correspond to the honest labeling), we say that a node $v \in V$ with parents $1 \leq v_1 < \ldots < v_d \leq N$ is* Green *if and only if $\ell'_v = H(v, \ell'_{v_1}, \ell'_{v_2}, \cdots, \ell'_{v_d})$. A node that is not* Green *is called a* Red *node.*

The ex-post facto graph $G' = \mathtt{EPFG}(G, \mathtt{trace}(A, H, \chi, R), \chi, \tau)$ can also be called the green-only subgraph i.e., the subgraph induced by all the nodes which are green in definition 9. For this subgraph, all nodes are locally consistent and thus their labels and the labels of their parents form a valid input-output pair to the random oracle. We now want to show that the Ex-Post-Facto Pebbling is legal for G'.

Challenge 3: Proving that the Ex-Post Facto Pebbling is Legal for the Ex-Post Facto Graph Recall that, in the context of regular data-independent memory hard functions, Alwen and Serbinenko [AS15] showed that the ex-post facto pebbling was a legal pebbling of the original graph G. In their context the graph G and the labels $\ell_1, \ldots, \ell_N$ are fixed a priori. In our context we want to show that the ex-post facto pebbling is a legal pebbling of the ex-post facto graph G'. However, because the ex-post facto graph is not fixed a priori we cannot directly apply Alwen and Serbinenko's [AS15] results to conclude that the ex-post facto pebbling is legal for G'. For example, we need to consider the ex-post facto labeling $\ell'_1, \ldots, \ell'_N$ when we define the pebbling. However, some of the key ideas from [AS15] can still be adapted. In particular, we define a predicate $\mathsf{need}(v, i)$ which indicates whether the (ex-post facto) label ℓ'_v will appear next as an input

before it is an output[3] before the prelabel x'_v of node v . We will have a pebble $v \in P_i$ on node v in round i if and only if its corresponding label ℓ'_v is needed during round i.

Definition 2 (Informal, see Definition 10). *Given an execution trace* $\texttt{trace}(A, H, \chi, R)$*, a string* $\ell' \in \{0,1\}^\lambda$*, and a particular round* i*, we define the predicate* $\mathsf{need}(v, i)$ *to be true if and only if the label* ℓ'_v *will next be used as input before it appears as the output to a random oracle query.*

Definition 3 (Informal, see Definition 11). *Given a DAG* $G = (V, E)$ *and an execution trace* $\texttt{trace}(A, H, \chi, R)$*, we define the Ex-Post-Facto Pebbling* $P' = \texttt{EPFP}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ *of the Ex-Post-Facto Graph* $P' = (P'_0, P'_1, \ldots, P'_t)$ *where* $P'_i = \{v \in V' | \mathsf{need}(v, i)\}$.

We then show that this Ex-Post-Facto Pebbling is, with high probability, a legal pebbling of our Ex-Post-Facto Graph by showing that any illegal moves necessitate some form of low-probability bad event (i.e. a hash collision or randomly guessing the output of a query).

Theorem 1 (Informal, see Theorem 4). *Except with negligible probability the Ex-Post-Facto Pebbling* $P' = \texttt{EPFP}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ *is a legal pebbling of the Ex-Post-Facto Graph* $G' = \texttt{EPFG}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$.

Challenge 4: Linking CMC and $\texttt{cc}(G')$ The next challenge is to argue that, except with negligible probability, $\texttt{cmc}(\texttt{trace}(A, H, \chi, R)) = \Omega(\texttt{cc}(P'))$ where P' is the Ex-Post-Facto Pebbling of our Ex-Post-Facto Graph. To prove this we use a similar extractor argument as Alwen and Serbinenko [AS15] to argue that, except with negligible probability, we have $|P'_i| \leq 4\,|\sigma_i|\,/\lambda$ for all rounds i. To prove this we show that there is an extractor which takes as input a hint of size $|\sigma_i| + |P'_i|\,(2\log_2 q + \log_2(\mathsf{indeg}(G)))$ and outputs $|P'_i|$ input/output pairs $\{(x'_v, \ell'_v)\}_{v \in P'_i}$ for the random oracle H without ever querying H on *any* of the inputs x'_v. Here, q denotes the number of random oracle queries that A makes and x'_v denotes the prelabel of node v i.e., given a node v with parents $v_1, \ldots, v_\delta$ in G and a fixed labeling $\ell'_1, \ldots, \ell'_N$ for each node in G we use $x'_v = \texttt{prelabel}(v) = (v, \chi, \ell'_{v_1}, \ldots, \ell'_{v_\delta})$ to denote the prelabel of node v.

Intuitively, the extractor works by simulating A from the initial state σ_i. For each node $v \in P'_i$ the hint tells us the index $h^b_v \leq q$ of the first random oracle query where label ℓ'_v appears as an input — this query x'_w will be the prelabel for another node w such that $(v, w) \in E(G')$. The hint also tells us the location h^w_v of ℓ'_v inside this prelabel — this requires $\log_2(\mathsf{indeg}(G))$ bits since w has at most $\mathsf{indeg}(G)$ parents. Thus, when the query x'_w is made the extractor can locate ℓ'_v and record the tuple $(v, _, \ell'_v)$. The extractor must also avoid querying the random oracle H on the prelabel x'_v which is tricky because x'_v is not known

[3] More precisely, if node v has parents $v_1, \ldots, v_\delta$ then we can define $x'_v = (v, \chi, \ell'_{v_1}, \ldots, \ell'_{v_\delta})$ as the prelabel of node v. Since $\ell'_v = H(x'_v)$ we would say that ℓ'_v is an output in round j if $x'_v \in Q_j$ is random oracle query during this round.

a priori. Thus, for each node $v \in P'_i$ the hint also includes the index h^a_v of the first time that the query x'_v is submitted to the random oracle. When A submits this query to H the extractor can update the tuple $(v, _, \ell'_v)$ to (v, x'_v, ℓ'_v) and simply respond with ℓ'_v without actually forwarding the query x'_v to H. Note that by definition of P'_i we know that the label ℓ'_v will appear as an input before it is output so we can safely assume that the tuple $(v, _, \ell'_v)$ was recorded when the query x'_v is submitted. If A submits the query x'_v to H in a future round the extractor can simply locate the tuple (v, x'_v, ℓ'_v) and respond with x'_v without querying H.

By simulating A the extractor will eventually be able to output $|P'_i|$ input/output pairs $\{(x'_v, \ell'_v)\}_{v \in P'_i}$ for the random oracle H without ever querying H on *any* of the inputs x'_v. Since H is a random oracle each string ℓ'_v is a uniformly random λ-bit string which cannot be compressed. In particular, given a hint of size $\leq |P'_i|\lambda - \lambda/2$ the probability of successfully extracting $|P'_i|$ input/output pairs without querying the random oracle on any of the inputs is at most $2^{-|P_i|\lambda + (|P'_i|\lambda - \lambda/2)} \leq 2^{-\lambda/2}$. It follows that (whp) we will have $|\sigma_i| \geq |P'_i|(\lambda - 2\log_2 q - \log_2 \mathsf{indeg}(G)) - \lambda/2 \geq |P'_i|\lambda/4$. The last inequality follows under the reasonable assumption that $\log_2 q + \log_2 \mathsf{indeg}(G) \leq \lambda/2$. By applying union bounds we can conclude that (whp) $|\sigma_i| \geq |P'_i|\lambda/4$ for every round i. It follows that, except with negligible probability, $\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) = \sum_i |\sigma_i| \geq \sum_i |P'_i|\lambda/4 \geq \mathtt{cc}(G')\lambda/4$.

Theorem 2 (Informal, see Theorem 6). *Except with negligible probability* $\mathsf{cmc}(\mathit{trace}(A, H, \chi, R)) \geq \mathtt{cc}(G')\frac{\lambda}{4}$*, where* $G' = \mathtt{EPFG}(G, \mathit{trace}(A, H, \chi, R), \chi, \tau)$ *is the Ex-Post-Facto Graph, and* τ *is the Merkle Tree root output by* A

Challenge 5: Lower Bounding the Pebbling Cost of the Ex-Post Facto Graph We have seen that $\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) \geq \mathtt{cc}(G')\lambda/4$ i.e., the cumulative memory complexity of A on input χ can be lower bounded by the cumulative pebbling complexity of the Ex-Post-Facto Graph G'. However, this lower bound is only meaningful if $\mathtt{cc}(G')$ is large. Unfortunately, it is possible that $\mathtt{cc}(G') \ll \mathtt{cc}(G)$ because G' is only a subgraph of G with all red (locally-inconsistent) nodes removed. Let S' denote the set of red nodes that were removed from G because they are not locally consistent i.e., $G' = G - S'$.

We first observe that if the number of red nodes $|S'|$ is too large then the probability $(1 - |S'|/N)^k$ that none of the k challenges are red will be negligible. Thus, we can safely assume that $|S'| \leq e$ for some parameter $e \leq N$ that will depend on the number of challenges k.

Our goal will be to find a graph G with the property that $\forall S' \subseteq V(G)$ of size $|S'| \leq e$ the subgraph $G' = G - S'$ has high cumulative pebbling complexity. We observe that depth-robust graphs satisfy this criteria. Thus, we suggest instantiating the Merkle Tree Proof framework with a highly depth-robust graph such as DRSample [ABH17]. We say that G is (e, d)-depth robust if for all $S \subseteq V(G)$ of size $|S| \leq e$ the graph $G - S$ still contains a directed path of length d. Alwen et al.[ABP17] showed that any (e, d)-depth-robust graph has cumulative pebbling complexity at least $\mathtt{cc}(G) > ed$. Furthermore, Alwen et

al. [ABH17,ABP17] showed how to construct (e, d)-depth robust graphs on N nodes with constant indegree for $e = \Omega(N/\log N)$ and $d = \Omega(N)$. It is easy to see that if G is $(2e, d)$-depth-robust then for any $S' \subseteq V(G)$ of size $|S'| \leq e$ the graph $G' = G - S'$ is still (e, d)-depth-robust and, therefore, $\mathtt{cc}(G - S') \geq ed$ for all $|S'| \leq e$. Thus, if G is $(2e, d)$-depth-robust we can conclude that either (1) there are at least $|S'| > e$ red nodes omitted from the $\mathtt{EPFG}$ G', or (2) $\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) \geq ed\lambda/4$ i.e., $\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) = \Omega(N^2\lambda/\log N)$ if G is $(\Omega(N/\log N), \Omega(N)))$-depth robust.

Challenge 6: Bounding the Probability of Lucky Challenges We just observed that when we instantiate G with a $(2e, d)$-depth-robust graph that either (1) we will have at least $|S'| > e$ red nodes, or (2) the cumulative memory complexity will be at least $ed\lambda/4$. It remains to show that we can tune the parameter k (number of challenges) to ensure that, except with negligible probability, the first outcome $|S'| > e$ will not occur. Let $x_{i,j}$ denote the jth random oracle query in round i and let $\tau_{i,j} = H(x_{i,j})$ denote the output of this query. We can interpret $\tau_{i,j}$ as the root of a Merkle Tree and and let $G'_{i,j} = G - S'_{i,j}$ denote the corresponding ex-post facto graph. Intuitively, we define a "lucky query" event $\textsc{LuckyQuery}_{i,j}$ which occurs if $c_1, \ldots, c_k \notin S'_{i,j}$ where $c_z = (H(\chi, z, \tau_{i,j}) \bmod N)$ i.e., none of the k challenges are red even though $|S'_{i,j}| > e$. If $x_{i,j}$ is a fresh random oracle query we can assume that (except with negligible probability) none of the random oracle queries $H(\chi, z, \tau_{i,j})$ were submitted before $x_{i,j}$. In this case the challenges $c_1, \ldots, c_k$ can be viewed as uniformly random and probability of the event $\textsc{LuckyQuery}_{i,j}$ is at most $(1 - e/N)^k$. If we set $k = (2N \ln 2\lambda)/e$ then the probability is at $2^{-\lambda}$. Union bounding over all q queries we can upper bound the probability that any lucky query occurs as $q2^{-\lambda}$. This leads to our primary result that the probability that an algorithm A produces a valid MHPoW with cumulative memory complexity below $\mathtt{cc}(G')\lambda/4$ is negligible.

Theorem 3 (Informal, see Theorem 7, Corollary 1 and 3). *Define the event* Success *that A produces a valid MHPoW that the verifier will accept then*

$$\Pr\left[\textsc{Success} \wedge \mathsf{cmc}\left(\mathtt{trace}(A, H, \chi, R)\right) < \frac{\lambda}{4}\left(\min_{S \subset V(G): |S| \leq e}(\mathtt{cc}(G - S))\right)\right] \leq \mathbf{negl}(\lambda) .$$

Furthermore, if G is $(2e, d)$-depth robust then

$$\Pr\left[\textsc{Success} \wedge \mathsf{cmc}\left(\mathtt{trace}(A, H, \chi, R)\right) < \frac{\lambda e d}{4}\right] \leq \mathbf{negl}(\lambda) .$$

When we instantiate the graph G with a $(\Omega(N/\log N), \Omega(N))$-depth robust graph like DRSample [ABH17] we can conclude that (WHP) any algorithm A must incur cumulative memory cost $\Omega(N^2\lambda/\log N)$ whenever it succeeds in generating a valid MHPoW.

Pebbling Attacks on Merkle Tree Memory Hard Proofs of Work Finally, in Section 7 we aim to lower bound the number of challenges k required to obtain a

secure MHPoW by developing pebbling attacks. Our basic observation is that the attacker can remove up to $e = N/k$ nodes from the graph G while ensuring that the probability $(1-(N/k)/N)^k = (1-1/k)^k = 1/e$ of avoiding detection remains large. Let $\mathtt{depth}(G, v)$ denote the length (number of nodes) in the longest directed path ending at node v in the DAG G. We first observe that if G' does not contain a directed path longer than d then G' can be pebbled in d parallel rounds by defining $P_i = \{v \in V(G') : \mathtt{depth}(G', v) \leq i\}$ for each round $i \leq d$. Therefore, if $\max_{v \in V(G')} \mathtt{depth}(\mathtt{G}', \mathtt{v}) \leq d$ then we have $\mathtt{cc}(G') \leq \sum_{i=1}^{d} |P_i| \leq dN$.

We say that a DAG G is (e, d)-reducible if it is not (e, d) depth-robust. In particular, G is (e, d)-reducible if there exists a subset $S \subseteq V(G)$ of size $|S| \leq e$ such that $\max_{v \in V(G) \setminus S} \mathtt{depth}(G - S, v) < d$ i.e., $G - S$ does not contain any directed path of length d. It is known that any DAG G with constant indegree is (e_i, d_i)-reducible with $d_i = N/2^i$ and $e_i = O(Ni/\log N)$ [AB16]. If G has constant indegree and $k \leq c \log N / \log\log N$ for a suitable constant c then we can find a set $S' \subseteq V(G)$ of size $|S'| \leq N/k = O(N \log\log N / \log N)$ such that $G' = G - S$ contains no directed path of length $N/\log^8 N$. It immediately follows that $\mathtt{cc}(G') \leq N^2/\log^8 N$, but the probability that the attacker avoids detection is constant. Thus, we must have $k = \Omega(\log N / \log\log N)$ if we want to obtain a MHPoW with cumulative memory complexity at least $\Omega(N^2\lambda/\log N)$.

We can develop improved attacks when G is instantiated with Argon2i which is known to be (e, d)-reducible with $d = O(N^3/e^3)$ for any parameter $e \geq N^{2/3}$ [BZ17]. We can further improve upon the previous attack $\mathtt{cc}(G') \leq d_i N$ by adapting the recursive pebbling strategy of Alwen et al. [ABP17] against a graph that is depth-reducible at on a range of values (e_i, d_i) i.e., we can find sets $S_0, S_1, \ldots$ such that $|S_i| \leq e_i$ and $G - S_i$ contains no directed path of length d_i. Alwen et al. [ABP17] showed that if the DAG G has $\mathsf{indeg}(G) \leq \delta$ and depth at most d_0 and is (e_1, d_1)-reducible with parameters e_1, d_2 such that $2d_1 N \leq e_1 d_0$ and S_1 is a depth reducing set of size $|S_1| \leq e_1$ such that $\mathsf{depth}(G - S_1) \leq d_1$ then, starting from an initially empty pebbling configuration, the cost of pebbling any target set $T \subseteq V(G)$ within at most $2d_0$ rounds is upper bounded by $\mathtt{cc}(G, T, 2d_0) \leq (4\delta + 4)e_1 d_0 + \frac{N}{e_1}\left(\max_{T' \subseteq V \setminus S_1 : |T'| \leq \delta e_1} \mathtt{cc}(G - S_1, T', 2d_1)\right)$. We trivially have $\mathtt{cc}(G - S_1, T', 2d_1) \leq 2d_1 N$, but if the subgraph $G - S_1$ is itself (e_2, d_2)-reducible with parameters e_2, d_2 such that $2d_2 N \leq e_2 d_1$ then the result can be applied applied recursively to further reduce costs. If the original graph is (e_i, d_i)-depth-reducible for a sequence $\{(e_i, d_i)\}$ such that $2d_{i+1} N \leq e_{i+1} d_i$ then we can continue recursing further to minimize pebbling costs. The key challenge is how to select sequence $\{(e_i, d_i)\}$ to minimize $\mathtt{cc}$. Alwen et al. [ABP17] showed how to do this for Argon2iA (an earlier version of Argon2i), and Blocki and Zhou [BZ17] showed how to do this for Argon2i. One key difference in our setting is that the first depth-reducing set $|S_0| = N/k$ is *actually* removed from the graph G to obtain the green only subgraph. Thus, we start with a graph of depth $d_0 = \mathsf{depth}(G - S_0)$ instead of $d_0 = N$. This yields a different cost optimization problem and allows us to achieve an even lower cumulative pebbling cost than [ABP17] or [BZ17].

2 Preliminaries

2.1 Parallel Random Oracle Model and Basic Facts

Given a oracle set $\mathbb{H}$, let A be any pROM algorithm on input x with random coins R, with parallel access to H sampled uniformly from $\mathbb{H}$.

Parallel Random Oracle Model Computation in the parallel random oracle model proceeds in rounds. An algorithm A is an oracle algorithm with the following behavior: In round $i+1$ the algorithm A receives as input the previous state $\sigma_i = (\tau_i, Q_i)$ as well as the responses $Y = H(Q_i) = [H(x_{i,1}), \cdots, H(x_{i,z_i})]$ to the random oracle queries $Q_i = \{x_{i,1}, \ldots, x_{i,z_i}\}$ made in the previous round. The algorithm A may then perform *arbitrary computation* [4] before outputting a new state τ_{i+1} and a new batch of random oracle queries Q_{i+1}. The initial state σ_0 encodes the initial input χ to the computation. We have $Q_0 = \{\}$. Computation ends when the algorithm A writes on a special output tape instead of making another batch of queries to the random oracle.

We say that a string s is an input at round i if for query $x_{i,j}$ we have $x_{i,j} = w_1 \| s \| w_2$ for some strings $w_1, w_2 \in \{0,1\}^*$ i.e., s is a substring of $x_{i,j}$.

The Oracle Prediction Game A useful game for bounding probabilities under the Random Oracle Model is the "Oracle Prediction Game". In this game, an algorithm P (which may be randomized) is given access to a ROM H and a hint h of size $|h|$. P may adaptively query H by submitting a query x_i (for some $1 \leq i \leq q$ and receiving the response $y_i = H(x_i)$. At the end of execution, P outputs a set of input output pairs $S = \{(a_1, b_1), \ldots, (a_d, b_d)\}$ such that none of the inputs were previously queried, i.e. $\forall (a_d, b_d) \in S, 1 \leq i \leq q : a_d \neq x_i$, yet all the pairs are correct, i.e. $\forall (a_d, b_d) \in S : H(a_d) = b_d$. The proof of this bound is well known; see, for example, [DKW11].

Lemma 1. *Given random coins R Let P be any predictor algorithm which receives a hint of size $|h|$, as described above. Then the probability (over the choice of H and R) that P outputs $|S|$ correct predictions of λ bits is at most $|h| 2^{-\lambda |S|}$.*

Fixing any input χ, random oracle H and random coins R for our algorithm A we obtain a sequence of states $\sigma_0, \sigma_1 \ldots, \sigma_t$. We call this sequence the execution trace and denote it as $\mathtt{trace}(A, H, \chi, R) = (\sigma_0, \ldots, \sigma_t)$ — note that $\mathtt{trace}(A, H, \chi, R) = (\sigma_0, \ldots, \sigma_t)$ is deterministically fixed once χ, H, R and A are specified. Following [AS15] we define the cumulative memory cost (CMC) of an execution trace as

Definition 4.

$$\mathsf{cmc}(\mathit{trace}(A, H, \chi, R)) = \sum_{i=1}^{t} |\sigma_i| \ .$$

[4] We do not constrain this computation in any way except that the algorithm cannot make random oracle queries in-between rounds. For example, A may run for exponential time before outputting the new state/batch of queries.

In particular, $\mathsf{cmc}(\mathtt{trace})$ is the sum of the size of the states during each round.

We use $Q_i(A, H, \chi, R)$ to denote the set of queries made during round i when we fix our algorithm A, random oracle H, input χ and random coins R. We use $x_{i,j}(A, H, \chi, R)$ to denote the jth query in $Q_i(A, H, \chi, R)$. It is also convenient to define $Q_{\leq i}(A, H, \chi, R) = \bigcup_{j=1}^{i} Q_i(A, H, \chi, R)$ as the set of all random oracle queries made in rounds 1 to i and $Q(A, H, \chi, R) = \bigcup_i Q_i(A, H, \chi, R)$ as the set of all random oracle queries made in any round. We say that a query $x_{i,j} \in Q_i(A, H, \chi, R)$ in round i is fresh if it has not been made previously i.e., $x_{i,j} \notin Q_{\leq i-1}(A, H, \chi, R)$. Note that if a query $x_{i,j}$ is fresh then $H(x_{i,j})$ can be viewed as a uniformly random string that is still yet to be sampled. We also define $q(A, H, \chi, R)$ as the total number of queries A makes i.e., $q = \sum_i |Q_i|(A, H, \chi, R)$. When A, H, χ, R are clear from the context we will often drop them from our notation and simply write Q_i, $x_{i,j}$, $Q_{\leq i}$ and Q respectively.

Bad Events

Given any pROM algorithm A we define the event $\textsc{Collision}(A, H, \chi, R)$ to denote the event that A produces a collision i.e., $\exists x_{i,j}, x_{k,\ell} \in Q(A, H, \chi, R)$ such that $x_{i,j} \neq x_{k,\ell}$ and $H(x_{i,j}) = H(x_{k,\ell})$.

We define the event $\textsc{BadOrder}_{i,j}(A, H, \chi, R)$ as the event that the jth query $x_{i,j} \in Q_i(A, H, \chi, R) \setminus Q_{\leq i-1}$ during round i is fresh and the string $H(x_{i,j})$ is a substring of some query in the set $Q_{\leq i}$.

We further define the event $\textsc{BadOrder}_i(A, H, \chi, R) = \bigcup_j \textsc{BadOrder}_{i,j}(A, H, \chi, R)$, i.e. that a bad order query occurs in round i. Finally, we define the event $\textsc{BadOrder}(A, H, \chi, R) = \bigcup_i \textsc{BadOrder}_i(A, H, \chi, R)$ as the event that there exists some round i where the event $\textsc{BadOrder}_i(A, H, \chi, R)$ occurs i.e., a bad order query occurs at any point.

It is well known that the probability of these events is bounded by $\Pr_H[\textsc{Collision}(A, H, \chi, R)] \leq \binom{q}{2} 2^{-\lambda}$ and $\Pr_H[\textsc{BadOrder}(A, H, \chi, R)] \leq q^2 n 2^{-\lambda}$. For completeness, we prove these bounds in Appendix A; see Lemmas 7 and 10.

2.2 Graph Notation

We use $G = (V = [N], E)$ to denote a directed acyclic graph (DAG) with $N = 2^n$ nodes. WLOG we assume that the vertices are already in topological order $1, 2, \ldots, N$ so that for any directed edge $(u, v) \in E$ we have $u < v$.

Basic Notation The indegree of node $v \in V$ is $\mathsf{indeg}(G, v) \doteq |\{u : (u, v) \in E|$ and the maximum indegree of a DAG G is $\mathsf{indeg}(G) = \max_{v \in V} \mathsf{indeg}(G, v)$. When G is clear from context we often drop G and write $\mathsf{indeg}(v)$. Given a node $v \in V$ we define $\mathsf{parents}(v, G) = \{u : (u, v)\}$ to be the set of all nodes u with outgoing edge to v. WLOG we assume that the set $\mathsf{parents}(v, G)$ is given in topological order i.e., $\mathtt{parents}(v) = (v_1, \ldots, v_{\mathsf{indeg}(v)})$ where $v_i < v_{i+1}$. When it is clear from context, we will drop the G and simply write $\mathsf{parents}(v)$. Given a DAG $G = (V, E)$ and a subset $S \subseteq V$ we define $G - S = (V - S, \{(u, v) \in E \text{ s.t. } u \notin S \wedge v \notin S\}$ to be the subgraph obtained by deleting all nodes in S and any incident edges.

We also define $G[S] = G - (V \setminus S)$ to be the subgraph obtained by deleting all nodes *except* those in S.

Graph Pebbling Let $G = (V, E)$ be a DAG and $T, S \subseteq V$ be node sets. Then a (legal) pebbling of G (with starting configuration S and target T) is a sequence $P = (P_0, \ldots, P_t)$ of subsets of V such that:

1. $P_0 = \emptyset$
2. Pebbles are added only when their parents already have a pebble at the end of the previous step. That is, $\forall i \in [t] \forall y \in P_i \setminus P_{i-1} \forall x \in \mathsf{parents}(y) x \in P_{i-1}$.
3. At some point every target node is pebbled (though not necessarily simultaneously). That is, $\forall x \in T \exists i \leq t x \in P_i$.

We call a pebbling of G complete if $S = \emptyset$ and T is the set of sink nodes of G.

We further define $cc(G, T, d)$ as the cost of the best parallel pebbling of G in at most d rounds which ends with pebbles on T.

The Labeling Game Given input χ and random oracle H and a DAG G we define labels $\ell_1, \ldots, \ell_N$ where the label of the first node is $\ell_1 = H(\chi, 1)$ and the label of each internal node $v \in [N]$ with $\mathsf{parents}(v) = (v_1, \ldots, v_k)$ is computed as $\ell_v = H(\chi, v, \ell_{v_1}, \ldots, \ell_{v_k})$ — note that ℓ_v can be computed once we have all of the labels $\ell_1, \ldots, \ell_{v-1}$.

In our setting an attacker may chose not to compute labels honestly. Fixing arbitrary labels $\ell'_1, \ldots, \ell'_N$ we say that node $v \in [N]$ with $\mathsf{parents}(v) = (v_1, \ldots, v_k)$ is locally consistent with respect to χ,H and $\ell'_1, \ldots, \ell'_N$ if and only if $\ell'_v = H(\chi, v, \ell'_{v_1}, \ldots, \ell'_{v_k})$.

Fixing a labeling $\vec{\ell'} = (\ell'_1, \ldots, \ell'_N)$ of our DAG $G = (V, E)$, an input χ and a node $v \in V$ with parents $\mathsf{parents}(v) = (v_1, \ldots, v_k)$ we define $\mathtt{prelabel}(v, \vec{\ell'}, G, \chi) = \left(\chi, v, \ell'_{v_1}, \ldots, \ell'_{v_k}\right)$. When G, $\vec{\ell'}$ and χ are clear from context we simply write $\mathtt{prelabel}(v)$.

We often find it useful to convert from a number to the binary string representing that number, especially when computing Merkle Trees. We use $\mathtt{bin}(v) = \sum_{i=1}^{k} v_i 2^{i-1}$ to denote the integer corresponding to the binary string v.

3 Merkle Tree Memory Hard Proofs of Work

3.1 Definition

This work is chiefly concerned with memory-hard proofs of work; a Memory Hard Proof of Work consists of two algorithms: an honest prover `Prove` and a verifier `Verify`. Note that we define Memory-Hard Proofs of work with respect to the random oracle model as Merkle Tree Memory-Hard Proofs of Work constructions are in the random oracle model. However, our definition 5 easily extends to standard model by requiring that the prover and verifier ignore the random oracle.

Definition 5. *A Memory Hard Proof of Work consists of two algorithms:* $\texttt{Prove}$ *and* $\texttt{Verify}$*. The prover* $\texttt{Prove}^H(\chi, N)$ *takes an input string* χ *and a hardness parameter* N*, and outputs a certificate* c*. The algorithm* $\texttt{Verify}^H(\chi, N, c')$ *takes both the original input string* χ *and hardness parameter, as well as a candidate certificate* c'*, and outputs a 1 (accept) or a 0 (reject). The Proof of Work must satisfy four properties*

1. *(Prover Efficiency) For all inputs* $\chi \in \{0,1\}^\lambda$*, hardness parameters* N*, random oracles* $H : \{0,1\}^* \to \{0,1\}^\lambda$*, and random coins* R *the algorithm* $\texttt{Prove}$ *has* $\texttt{cmc}(\textit{trace}(\texttt{Prove}, H, \chi, R)) \leq N^2\lambda$ *and terminates after at most* $O(N)$ *rounds.*
2. *(Verifier Efficiency) We say that the Memory Hard Proof of Work is* $k(N,\lambda)$ *efficient if, for all inputs* $\chi \in \{0,1\}^\lambda$*, hardness parameters* N*, random oracles* $H : \{0,1\}^* \to \{0,1\}^\lambda$*, random coins* R *and candidate certificates* c' *the algorithm* $\texttt{Verify}^H(\chi, Nc'; R)$ *terminates after* $k(N,\lambda)$ *rounds. We will typically require that* $k(N,\lambda) = o(N)$*.*
3. *(Perfect Completeness) For all inputs* χ *and hardness parameters* N *we have* $\Pr\left[\texttt{Verify}^H\left(\chi, N, \texttt{Prove}^H(\chi, N)\right) = 1\right] = 1$ *where the randomness is taken over the selection of the random oracle* H *as well as any random coins of* $\texttt{Prove}$ *or* $\texttt{Verify}$*.*
4. *(Soundness) We say that the Proof of Work is* $(\epsilon(N,\lambda), C(N,\lambda))$*-sound if for all inputs* χ*, hardness parameters* N*, random oracles* $H : \{0,1\}^* \to \{0,1\}^\lambda$*, random coins* R*, and adversaries* $A^H(\chi, N; R)$ *which output a candidate certificate* c'*,* $\Pr\left[\texttt{Verify}^H\left(\chi, N, \mathcal{A}^H(\chi, N)\right) = 1 \wedge \texttt{cmc}(\textit{trace}(A, H, \chi, R) \leq C(N,\lambda))\right] \leq \epsilon(N,\lambda)$

3.2 Merkle Tree Proof Framework

The following is a folklore construction of a memory-hard proof of work, which is similar to [BK16]. Fix a random oracle H and a family of DAGs $\{G_N\}_{n\geq 1}$ where G_N is a DAG with N nodes $V_N = [N]$ and constant indegree. When N is clear from context we will often write G instead of G_N to simplify notation. We will also assume that $N = 2^n$ is a power of 2 to simplify presentation. The construction also utilizes a tunable parameter k (number of challenges). Intuitively, increasing k will decrease verifier efficiency, but will boost soundness.

We define a related event, Success. For a fixed graph G and random oracle H and given an input string χ, random coins R, a number of challenge rounds k, and any algorithm A, we say that $\textsc{Success}(G, H, \chi, R, A, k)$ occurs if and only $V(G, H, \chi, R, k, A(G, H, \chi, R, k))$ accepts, i.e. A produces a verified certificate. We say that A succeeds with probability α if $\Pr_H[\textsc{Success}(G, H, \chi, R, A, k)] \geq \alpha$, where the probability is taken over the choice of random oracle.

Given an input string χ, random coins R, and a number of challenge rounds k, the honest prover $\texttt{Prove}^H(\chi, N, k)$ proceeds as follows:

1. $\texttt{Prove}^H(\chi, N, k)$ computes the label each node $v \in V(G)$ in topological order. The label of the source node 1 is $\ell_1 = H(\chi, 1)$. One we have computed labels

$\ell_1, \ldots, \ell_{v-1}$ we can compute ℓ_v by querying $\ell_v = H(\chi, v, \ell_{v_1}, \ldots, \ell_{v_k})$ where $\ell_{v_1}, \ldots, \ell_{v_k} = \mathsf{parents}(v, G_N)$ denote the parents of node v.

2. $\mathtt{Prove}^H(\chi, N, k)$ computes a Merkle Tree commitment to this labeling, producing root τ. In particular, for each $v \in \{0,1\}^n$ we write $\tau_v = \ell_{1+bin(v)}$ and for $v \in \{0,1\}^{\leq n}$ define $\tau_v = H(\chi, \tau_{v\|0}, \tau_{v\|1})$ and the root is $\tau = H(\chi, \tau_0, \tau_1)$.
3. For each $i \in [1,k]$, $\mathtt{Prove}^H(\chi, N, k)$ identifies the node $c_i = (H(\chi, i, \tau) \mod N)$ as the ith challenge, then:
   (a) $\mathtt{Prove}$ computes a Merkle Tree Reveal $\mu_i = \mathtt{MerkleTreeReveal}(\mathsf{bin}(c_i))$ for the label ℓ_{c_i+1} where we define $\mathtt{MerkleTreeReveal}(x = (x_1, \ldots, x_n)) = \left(\tau_x, \{\tau_{x_1 x_2 \ldots x_{j-1}\overline{x}_j}\}_{j=1}^n\right)$ where $x_1, \ldots, x_n$ denote the bits of $x \in \{0,1\}^n$ and $\overline{x}_j$ denotes the bit flip of x_j,
   (b) Similarly, for each parent $v \in \mathsf{parents}(c_i + 1)$, $\mathtt{Prove}$ computes a Merkle Tree Reveal for the label $\ell_v = \tau_{\mathsf{bin}(v-1)}$ i.e., $\mu_{i,v} = \mathtt{MerkleTreeReveal}(\mathsf{bin}(v-1))$. Let $L_i = \{(\ell_v, \mu_{i,v})\}_{v \in \mathsf{parents}(c_i+1)}$.
4. $\mathtt{Prove}^H(\chi, N, k)$ can then output the certificate $c = \tau, \{(\ell_1, \mu_1, L_1), \ldots, (\ell_k, \mu_k, L_k)\}$.

The corresponding verifier $\mathtt{Verify}^H(\chi, R, c', k)$ proceeds as follows:

1. $\mathtt{Verify}$ identifies τ from the candidate certificate c', and computes all the challenges $c'_i = (H(\chi, i, \tau) \mod N)$.
2. For each challenge i, $\mathtt{Verify}$ separates the response into $L = \{(\ell_i, \mu_i, \{\ell_j, \mu_j\}_{j \in \mathsf{parents}(c_i+1)})\}_{i \in [1,k]}$, and runs MerkleTreeReveal$(\chi, i, \ell_i, \mu_i, \tau)$ on each of the reveals. If any fails, $\mathtt{Verify}$ rejects.
3. For each challenge, $\mathtt{Verify}$ checks that $\ell_v = H(\chi, v, \ell_{v_1}, \ldots, \ell_{v_k}$ holds for the labels the prover committed to. If this does not hold for any challenge, $\mathtt{Verify}$ rejects.
4. If c' passes all the above checks, $\mathtt{Verify}$ accepts.

Efficiency: The honest prover runs in sequential time $O(N + k \log N)$ and has cumulative memory complexity at most $\mathsf{cmc}(\mathtt{Prove}^H(\chi, N, k)) \in O(N^2\lambda + kN\lambda)$ i.e., the honest evaluation algorithm allocates $O(N\lambda)$ space to store $O(N)$ labels and keeps these labels in memory for the duration of the compuation. Assuming that we set $k = o(N/\log N)$ the sequential running time is at most $O(N)$ and the cumulative memory complexity is at most $\mathsf{cmc}(\mathtt{Prove}^H(\chi, N, k)) \in O(N^2\lambda)$. The size of the proof is $O(k\lambda \log N)$ since we must reveal $O(k)$ labels and each Merkle Tree Reveal requires $O(\lambda \log N)$ bits. The running time of the verifier is $O(k \log N)$ since the verifier must make $O(k \log N)$ queries to the hash function to verify the Merkle Tree openings and check that each revealed label is locally consistent. We will assume that $k \ll N/\log N$. Thus, the size of the proof is $o(N\lambda)$ and the running time of the verifier is $o(N)$.

Correctness: It is readily apparent that an honest prover P's certificate will always be accepted by a verifier V, so this procedure passes the correctness requirements. What remains is to prove its soundness.

4 Security Proof

In this section, we prove Theorem 6, which states that, as long as no bad events occur, the cmc of any attacker which computes a valid Memory-Hard Merkle Tree Proofs certificate is lower bounded by the mimimum cc of the Ex-Post-Facto Graph.

4.1 Tracing a Merkle Tree

Merkle Tree Commitments We utilize Merkle Tree commitments in our constructions. Given strings $x_1, \ldots, x_{2^k}$, a hash function H, and a salt s, we define the following labeling: for each string $v \in \{0,1\}^k$ we have $\ell'_v = x_{1+\mathtt{bin}(v)}$ where $\mathtt{bin}(v) = \sum_{i=1}^{k} v_i 2^{i-1}$ denotes the integer corresponding to the binary string v. For each $i \leq k$ and each string $v \in \{0,1\}^i$ we define $\ell_v = H(s, \ell'_{v\|0}, \ell'_{v\|1})$. Finally, the Merkle Tree commitment to $x_1, \ldots, x_{2^m}$ is given by the root label $\ell'_\epsilon = H(s, \ell'_0, \ell'_1)$.

Tracing a Merkle Tree In the random oracle model we can often reconstruct the entire Merkle Tree given only the root commitment ℓ'_ϵ and the salt s. In particular, suppose that the database $D = \{(x_1, y_1), \ldots, (x_q, y_q)\}$ encodes all known input/output pairs for our random oracle. We can attempt to reconstruct the Merkle Tree by looking for any $(x_i, y_i) \in D$ such that X_i takes the form $x_i = s\|w_0\|w_1$ for arbitrary strings $w_0, w_1 \in \{0,1\}^\lambda$ and $y_i = \ell'_\epsilon$. Note that as long as there are no collisions in D that there will be *at most* one such pair (x_i, y_i) in D. Once we find $(x_i, y_i) \in D$ with $y_i = \ell'_\epsilon$ and $x_i = s\|w_0\|w_1$ we can recursively trace the left (resp. right) subtree by setting $\ell'_0 = w_0$ (resp. $\ell'_1 = w_1$) and searching for $(x_j, y_j) \in D$ such that $y_j = \ell'_0$ (resp. $y_j = \ell'_1$) and x_j takes the form $x_j = s\|w'_0\|w'_1$ for arbitrary strings $w'_0, w'_1 \in \{0,1\}^\lambda$. It is of course possible that there is no pair $(x_j, y_j) \in D$ with $y_j = \ell'_0$ and x_j of the form $x_j = s\|w'_0\|w'_1$ for arbitrary strings $w'_0, w'_1 \in \{0,1\}^\lambda$. In this case we can simply set $\lambda_{0\|x} = \perp$ for all $x \in \{0,1\}^{\leq m-1}$ to indicate that it is not possible to further expand the left-subtree i.e., all the other nodes in this subtree have label $\perp$.

We use $\mathtt{MT}(\tau, D, s, k)$ to denote the recursive procedure which attempts to trace a Merkle Tree of depth k with root value $\ell_\epsilon = \tau$. In particular, the procedure will output the associated label $\ell'_v \in \{0,1\}^\lambda$ for all nodes $v \in \{0,1\}^{\leq k}$ in the traced Merkle Tree. It is possible that $\ell'_v = \perp$ for some nodes $v \in \{0,1\}^k$ if the Merkle Tree is incomplete. However, as long as the database D contains no collisions the output of $\mathtt{MT}(\tau, D, s, k)$ is uniquely defined.

Definition 6. *Given an execution trace* $\mathtt{trace}(A, H, \chi, R)$ *we can define the database* $D(\mathtt{trace}(A, H, \chi, R))$ *as the set of all queries made within the trace. That is,* $D(\mathtt{trace}(A, H, \chi, R)) = \bigcup_i Q_i(A, H, \chi, R)$.

Definition 7. *Given a directed acyclic graph* G *with nodes* $V = [N]$*, an input* χ*, and a fixed labeling* $\ell'_1, \ldots, \ell'_N \in \{0,1\}^\lambda \cup \{\perp\}$ *(which may or may not correspond to the honest labeling), we say that a node* $v \in V$ *with parents* $1 \leq v_1 < \ldots <$

$v_d \leq N$ *is* Green *if and only if (1)* $\ell'_v \neq \perp$*, (2)* $\ell'_{v_i} \neq \perp$ *for all* $i \leq d$ *and (3)* $\ell'_v = H(\chi, v, \ell'_{v_1}, \ell'_{v_2}, \cdots, \ell'_{v_d})$*. A node that is not* Green *is called a* Red *node.*

Recall that given a database D of random oracle output queries, a DAG G with $N = 2^n$ nodes, and a candidate Merkle Tree root value τ we can use $\texttt{MT}(\tau, D(\texttt{trace}(A, H, \chi, R)), \chi, n)$ (where χ, our input, is also used as the salt) to trace labels ℓ'_v for each node $v \in \{0,1\}^{\leq n}$ in the Merkle Tree. As long as the database D contains no collisions the values ℓ'_v are uniquely determined once we fix τ, D, χ and n. In this case we can obtain a fixed labeling for our DAG G by setting $\ell'_i = \ell'_{\texttt{bin}^{-1}(i-1,n)}$ for each $i \in [N]$ where $\texttt{bin}^{-1}(i-1, n)$ denotes the binary encoding of $i - 1$ as a n-bit string.

Definition 8. *Given a graph* G*, execution trace* $\texttt{trace}(A, H, \chi, R)$*, corresponding initial input* χ*, and a Merkle Tree root value* τ*, when* $D = D(\texttt{trace}(A, H, \chi, R))$ *we call* $\ell'_1, \ldots, \ell'_n$ *(as traced above) the Ex-Post-Facto Labeling of* G *and denote it by* $\texttt{EPFL}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$*.*

Thus, assuming there are no collisions we can determine which nodes in a graph G are *Green*/*Red* given only $\texttt{trace}(A, H, \chi, R)$, χ and τ. It will be convenient to let $S_{Red}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ denote the set of nodes that are red.

Definition 9. *Given a graph* G*, execution trace* $\texttt{trace}(A, H, \chi, R)$*, corresponding initial input* χ*, and a certificate rooted at* τ*, we define the Ex-Post-Facto Graph* G' *to be the subgraph of* G *induced by the nodes which are green. That is,* $G' = G - S_{RED}$*. We denote this by* $G' = \texttt{EPFG}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$

Tracing an Ex-Post-Facto Pebbling Therefore, for any given graph G, execution trace $\texttt{trace}(A, H, \chi, R)$, corresponding initial input χ, and candidate Merkle Tree root τ, we begin by tracing the Merkle Tree labeling with $\texttt{MT}(\tau, D, \chi, n)$. This lets us find the Ex-Post-Facto Labeling ℓ'_v for each vertex v. Then, we find the Ex-Post-Facto Graph G' , which contains only *Green* nodes with respect to the ex post facto labeling $\ell'_1, \ldots, \ell'_N$.

We now use our Ex-Post-Facto Labeling to define a pebbling of our Ex-Post-Facto Graph. In particular, for each step i in our $\texttt{trace}$ and each vertex v in G' we consider whether the corresponding label ℓ'_v will be "needed" as input to a random oracle query before the next round where this label will appear as an output. Intuitively, during round i we keep keep pebbles on the set P'_i of nodes that are "needed." Given a node v with parents $v_1, \ldots, v_\delta$ it will be convenient to let $\texttt{prelabel}(v) = (v, \chi, \ell'_{v_1}, \ldots, \ell_{v_\delta})$ denote the prelabel of node v i.e., the random oracle query such that $H(x'_v) = \ell'_v$.

Definition 10. *Given an execution trace* $\texttt{trace}(A, H, \chi, R)$*, an Ex-Post-Facto Labeling* $L' = \ell'_1, \ldots, \ell'_N$*, a vertex* $v \in V(G')$*, and a particular round* i*, we define the predicate* $\texttt{need}(\texttt{trace}(A, H, \chi, R), L', v, i)$ *to be true if and only if the label* ℓ'_v *will next be used as input to a query before being output by a query. That is,* $\texttt{need}(\texttt{trace}(A, H, \chi, R), L', v, i) = 1$ *if and only if (1) there exists a query* $x_{j,b} \in Q_j(A, H, \chi, R)$ *such that* $i \leq j$*, and* $x_{j,b} = \texttt{prelabel}(w)$ *for some node*

w such that $(v, w) \in E(G')$*, but (2) there is no* $x_{k,l} \in Q_k(A, H, \chi, R)$ *for any* $i \leq k < j$ *such that* $x_{k,l} = \texttt{prelabel}(v)$*. For simplicity we will write* $\mathsf{need}(v, i)$ *instead of* $\mathsf{need}(\texttt{trace}(A, H, \chi, R), L', v, i)$ *When* $\texttt{trace}(A, H, \chi, R)$ *and* L' *are clear from context.*

We are now ready to define the Ex-Post-Facto Pebbling of the Ex-Post-Facto Graph.

Definition 11. *Given a DAG* $G = (V, E)$*, an execution trace* $\texttt{trace}(A, H, \chi, R)$*, the corresponding initial input* χ*, and a Merkle Tree root* $\tau \in \{0,1\}^\lambda$ *we define the Ex-Post-Facto Pebbling* $P' = \texttt{EPFP}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ *of the Ex-Post-Facto Graph* $G' = \texttt{EPFG}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ *as follows:* $P' = (P'_0, P'_1, \ldots, P'_t)$ *where* $P'_i = \{v \in V' \mid \mathsf{need}(\texttt{trace}(A, H, \chi, R), L', v, i) = 1\}$ *where* $L' = (\ell'_1, \ldots, \ell'_n) = \texttt{EPFL}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ *denotes the Ex-Post-Facto Labeling of the Ex-Post-Facto Graph* $G' = (V', E')$*.*

Consider a given graph G, algorithm A, ROM H, input string χ, and random coins R. Recall that a node v with parents $v_1, \ldots, v_d$ is *Green* if it is internally consistent: it has a label ℓ'_v which can be traced from the Merkle Tree by $\texttt{EPFL}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ and that label is the hash of its parents' labels, i.e. $\ell'_v = H(\chi, v, \ell'_{v_1}, \ldots, \ell'_{v_d})$. Intuitively, if v is *Green* then A must have found ℓ'_v by querying $H(\chi, v, \ell'_{v_1}, \ldots, \ell'_{v_d})$. However, this intuition is not guaranteed to be correct. It is possible, albeit highly unlikely, that A simply guessed ℓ'_v, yet this guess was by sheer chance internally consistent. We call this MisColoring the node v, and give the following formal definition.

Definition 12. *Fix a graph* G*, algorithm* A*, input string* χ*, random coins* R *and random oracle* H*. Let* $tr = \texttt{trace}(A, H, \chi, R)$ *be the execution trace when running* A *on* χ *and let* $\vec{\ell'} = \texttt{EPFL}(G, tr, \chi, \tau)$ *be the Ex-Post-Facto Labeling of* G *where* τ *is the Merkle Tree root output by* A*. Given a vertex* $v \in V(G)$*, we define the event* $\textsc{MisColor}_v(G, A, H, \chi, R, \tau)$ *as the event that* v *is* Green *with respect to* $\vec{\ell'}$*,* χ*, and* G *but the string* $\texttt{prelabel}(v, \vec{\ell'}, G, \chi)$ *has never been queried i.e.,* $x_{i,j} \neq \texttt{prelabel}(v, \vec{\ell'}, G, \chi)$ *for all* i, j*. In this case we say that the vertex* v *was miscolored. We further define the event that any vertex is miscolored as*

$$\textsc{MisColor}(G, A, H, \chi, R, \tau) = \bigcup_v \textsc{MisColor}_v(G, A, H, \chi, R, \tau) \ .$$

Lemma 2. *Let* G *be a graph;* $\texttt{trace}(A, H, \chi, R)$ *be an execution trace of a pROM algorithm* A *for ROM* $H : \{0,1\}^n \to 2^\lambda$*, input* χ*, and random coins* R*;* χ *be the corresponding input to* A*; and* τ *be the Merkle Tree commitment generated. For all inputs* χ *and random coins* R *we have* $\Pr_H[\textsc{MisColor}(G, A, H, \chi, R, \tau)] \leq |V(G)|2^{-\lambda}$*, where the probability is taken over the selection of the random oracle* H*.*

Proof. Consider the Ex-Post-Facto Labeling of G given by $\texttt{EPFL}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$. First we will bound the probability of $\textsc{MisColor}_v(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ for any vertex in $v \in V(G)$. For v to be *Green* we must have (1) a fixed label ℓ'_v,

(2) for parents of v $1 \leq v_1 < \ldots < v_d$, we must have fixed labels ℓ'_{v_i} for all $i \leq d$, and (3) $\ell'_v = H(\chi, v, \ell'_{v_1}, \ell'_{v_2}, \cdots, \ell'_{v_d})$. We know by the definition of MisColor that no query made in the trace outputs ℓ'_v, so the query $H(\chi, v, \ell'_{v_1}, \ell'_{v_2}, \cdots, \ell'_{v_d})$ of the prelabel of ℓ'_v must be fresh, and can be viewed as a uniformly random string that has yet to be sampled. Yet this uniformly random string of λ bits is exactly equal to the fixed string ℓ'_v. This occurs with chance $2^{-\lambda}$. Thus, $\Pr_H[\textsc{MisColor}_v(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)] = 2^{-\lambda}$

Now, recall that $\textsc{MisColor}(G, A, H, \chi, R, \tau) = \bigcup_v \textsc{MisColor}_v(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$. Thus, by a simple union bound we have

$$\Pr_H[\textsc{MisColor}(.)] \leq \textstyle\sum_{v \in V(G)} \Pr_H[\textsc{MisColor}_v(.)] \leq \displaystyle\sum_{v \in V(G)} 2^{-\lambda} \leq |V(G)| 2^{-\lambda} \ ,$$

where $\textsc{MisColor}(.) = \textsc{MisColor}(G, G, A, H, \chi, R, \tau)$. □

Theorem 4. *If the bad events* $\textsc{Collision}(A, H, \chi, R)$, $\textsc{BadOrder}(A, H, \chi, R)$, *and* $\textsc{MisColor}(G, A, H, \chi, R, \tau), \chi, \tau)$ *never occur, then* $P' = \texttt{EPFP}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ *is a legal pebbling of* $G' = \texttt{EPFG}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$.

Proof. Let $\texttt{tr} = \texttt{trace}(A, H, \chi, R)$ denote the execution trace when we run algorithm A on input χ with fixed random coins R and random oracle H and let $D = D(\texttt{trace}(A, H, \chi, R))$ denote the database of random oracle queries. We can assume that D has no collisions since the event $\textsc{Collision}(A, H, x, R)$ does not occur. Recall that if D does not have any collisions, we are able to trace a unique Ex-Post-Facto Labeling $L' = \ell'_1, \ldots, \ell'_n = \texttt{EPFL}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$, a unique Ex-Post-Facto Graph $G' = \texttt{EPFG}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$, and a unique Ex-Post-Facto Pebbling $P' = \texttt{EPFP}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ using definitions 8 9 and 11.

We will first argue that all nodes in G' must be pebbled at some step. Intuitively, we know that a node $v \in V(G') \subseteq \{0,1\}^n$ can only be *Green* if its label $\ell'_v \neq \perp$ was traced by the procedure $\texttt{MT}(\tau, D(\texttt{trace}(A, H, \chi, R)), \chi, n)$. Let $v = v' \| b$ where $b \in \{0,1\}$ and $v' \in \{0,1\}^{n-1}$. By definition of MT since $\ell'_v \neq \perp$ there must have been some random oracle query of the form $H(\chi, \ell_{v' \| 0}, \ell_{v' \| 1})$. Therefore, we have $\texttt{need}(\texttt{trace}(A, H, \chi, R), L', v, i), \ell', i) = 1$ (and $v \in P'_i$) for some round $i \geq 0$ since $\ell_{v' \| b}$ appeared as an input for some random oracle query. In particular, every node $v \in V(G')$ is pebbled at some point i.e., pebbling condition 3 is satisfied.

Suppose for contradiction that $P'_0 \neq \emptyset$ then there is some node $v \in P'_0$. Since the event MisColor does not occur ℓ'_v must be the output of some future random oracle query $x_{k,l}$ — let k be the earliest round in which ℓ'_v appears as output. However, by definition of P' we have $\texttt{need}(\texttt{trace}(A, H, \chi, R), L', v, 0) = 1$ which means that ℓ'_v appears as input in some future round before round k when ℓ'_v appears as output for the first time. This violates our assumption that the event $\textsc{BadOrder}(A, H, \chi, R)$ never occurs. Thus, pebbling condition 1 is satisfied.

Now suppose for contradiction that pebbling condition 2 is not satisfied i.e., for some round b we have $v \in P'_{b+1} \setminus P'_b$ even though $\mathsf{parents}(v) \not\subset P'_b$. Let b be the earliest round in which this occurs. Because $v \in P'_{b+1}$ and $v \notin P'_b$ we have $\mathsf{need}\left(\mathtt{trace}(A, H, \chi, R), L', v, b+1\right) = 1$ and $\mathsf{need}\left(\mathtt{trace}(A, H, \chi, R), L', v, b\right) = 1$ respectively which means that ℓ'_v must be the output of a random oracle query made during round b. In particular, since the event $\textsc{Collision}(A, H, x, R)$ does not occur we must have $x_{b,j} = \mathtt{prelabel}(v, \vec{\ell}', G, \chi)$ for some query made during round b. However, for any node $u \in \mathsf{parents}(v, G')$ the label ℓ'_u is a substring of $x_{b,j}$. It follows that $\mathsf{need}\left(\mathtt{trace}(A, H, \chi, R), L', u, b\right) = 1$ for all $u \in \mathsf{parents}(v, G')$. Therefore, $u \in P'_b$ for all $u \in \mathsf{parents}(v, G')$ contradicting our assumption. Thus, pebbling condition 2 must be satisfied. □

4.2 Extracting Input/Output Pairs from the Random Oracle

In the previous section we argued that the ex-post facto pebbling of the ex-post facto graph must be legal (as long as certain highly unlikely "bad" events do not occur). We now argue that during each round of our execution trace the size of the state σ_i is lower bounded by $|P'_i|$ i.e., the number of pebbles in the ex-post facto pebbling. We use an extractor argument to prove this. The extractor will output $|P'_i|$ input/output pairs for the random oracle without querying the random oracle at this points. Intuitively, the random oracle outputs will correspond to the ex-post facto label ℓ'_v of each node $v \in P'_i$. Recall that, by definition, for any vertex $v \in P'_i \in= \mathtt{EPFP}(G, \mathtt{trace}(A, H, \chi, R), \chi, \tau)$ we have $\mathsf{need}(v, i)$ i.e., the ex-post facto label ℓ'_v will appear as an input to a future random oracle query *before* it is output. Thus, it will be possible for the extractor to extract each ex-post facto label ℓ'_v by simulating our algorithm A starting from the initial state σ_i given a small hint about which query contains each label ℓ'_v as part of the input.

Defining the Extractor Intuition: Given any intermediate state $\sigma_i = (\tau_i, Q_i)$ of A, access to the random oracle H, an input χ, and random coins R, we can simulate A from this initial state. The goal of the extractor is to output $\left(\mathtt{prelabel}(v, \vec{\ell}', G, \chi), \ell'_v\right)$ for each node $v \in P'_i$ without every querying $\mathtt{prelabel}(v, \vec{\ell}', G, \chi)$. To simulate A we will need the *responses* to the random oracle queries Q_i. We can obtain these responses simply by submitting each query $x_{i,j} \in Q_i$ to the random oracle. However, if $x_{i,j} = \mathtt{prelabel}(v, \vec{\ell}', G, \chi)$ for some node $v \in P'_i$ then we must *not* submit this query. However, if we have already extracted ℓ'_v then it will be possible to find the output $H(x_{i,j})$ without actually making this query. Thus, for each node $v \in P'_i$ our hint will include the index of the first query where ℓ'_v appears as part of a query input as well as the index of the first query where the input is $\mathtt{prelabel}(v, \vec{\ell}', G, \chi)$ — if such a query exists. Note that, by definition, ℓ'_v will appear first as input so if we do encounter a future query $\mathtt{prelabel}(v, \vec{\ell}', G, \chi)$ we will be able to respond to the query without needing to query the random oracle. If the query

$\texttt{prelabel}(v, \vec{\ell'}, G, \chi)$ never occurs in our simulation of A starting at round i we can still extract $\texttt{prelabel}(v, \vec{\ell'}, G, \chi)$ by subsequently simulating A from the initial state σ_0. Note that the state σ_0 (as well as the random coins R) are entirely independent of the random oracle H and thus does not count as part of our random oracle dependent hint.

RO Dependent Hint In addition to the initial state σ_i we will give our extractor a hint $h_i = \{(h_v^b, h_v^w, h_v^a)\}_{v \in P_i'}$. In particular, for each node v in P_i' the value $h_v^b \leq q$ is the index of the first query where ℓ_v' will appear as part of the input and $h_v^w \leq \mathsf{indeg}(G)$ denotes which portion of this query is ℓ_v'. Similarly, $h_v^a \leq q$ denotes the index of the first query where the input is $\texttt{prelabel}(v, \vec{\ell'}, G, \chi)$ i.e., the query whose output would be ℓ_v'. Note that it is possible that this query occurs after the extractor finishes simulating A from the intermediate state σ_i and begins simulating A from the initial state σ_0 until we reach the state σ_i. The total number of random oracle queries throughout the simulation is at most q so we are still guaranteed that $h_v^a \leq q$. We also note that $h_v^a > h_v^b$ since the label ℓ_v' will appear as input before it appears as output.

Formal Description of the Extractor Given the intermediate state $\sigma_i = (\tau_i, Q_i)$ of A the extractor E first examines each query $x_{i,j} \in Q_i$. If $x_{i,j}$ is the h_v^b-th query made for any h_v^b in the hint, then ℓ_v' is the h_v^w-th parent used in this query, given by $x_{i,j}[|\chi| + \log_2(N) + h_v^w \cdot \lambda, |\chi| + \log_2(N) + (h_v^w + 1) \cdot \lambda - 1]$. E extracts this value, infers v by examining the structure of G to see which node is the h_v^w-th parent, and records the tuple $(v, _, \ell_v')$ to indicate that it has extracted the ex-post facto label ℓ_v', but does not yet know $\texttt{prelabel}(v)$. Now, if $x_{i,j}$ is the h_v^a-th query made for any h_v^a in the hint, E can find the tuple $(v, _, \ell_v')$ and update this record to $(v, x_{i,j}, \ell_v')$ to show that it knows both the label and prelabel for the node v. Finally, E checks whether there is some tuple of the form $(u, x_{i,j}, \ell_u')$ in its records. If so, it does not make this query and instead sets $y_{i,j} = \ell_u'$. Otherwise, E can safely make the random oracle query and set $y_{i,j} = H(x_{i,j})$. Now that E has the random oracle responses Y_i it will continue to simulate A to get the next state $\sigma_{i+1} = (\tau_{i+1}, Q_{i+1})$.

In general when E obtains the state $\sigma_r = (\tau_r, Q_r)$ for some round $r > i$ it proceeds in the exact same manner. In particular, for each query $x_{r,j} \in Q_r$, E checks if $x_{r,j}$ is the h_v^b-th query made for any h_v^b in the hint, recording the tuple $(v, _, \ell_v')$ if so. E checks if $x_{r,j}$ is the h_v^a-th query made for any h_v^a in the hint, updating the record $(v, _, \ell_v')$ to $(v, x_{r,j}, \ell_v')$ if so. Finally, we check whether there is a tuple of the form $(u, x_{r,j}, \ell_u')$ in our records. If so, we do not make this query and instead set $y_{r,j} = \ell_u'$. Otherwise, we make the random oracle query and set $y_{r,j} = H(x_{r,j})$. Our extractor continues simulating A in this manner until A terminates. However, it is of course possible that we still have some tuple in our record of the form $(u, _, \ell_u')$ if no query of the form $x_{k,l} = \texttt{prelabel}(u)$ has been made. In this case, the extractor will continue to simulate A, beginning from the initial state $\sigma_0 = (\chi, \emptyset)$ until it reaches state σ_i at round i. During this simulation during each round $r \leq i$ we continue to check each query $x_{r,j} \in Q_r$ to see if $x_{r,j}$ is the h_j^a-th query made for any h_v^a in the hint. When this occurs

we update the tuple $(v, _, \ell'_v)$ to $(v, x_{r,j}, \ell'_v)$. We also double check every query $x_{r,j}$ to see if some tuple of the form $(u, x_{r,j}, \ell'_u)$ was previously recorded so that we can avoid querying $\texttt{prelabel}(u)$ for any node $u \in P'_i$. Finally, the extractor will locate all of the tuples of the form (u, w_u, ℓ'_u) and output the random oracle input/output pair (w_u, ℓ'_u).

We use E to denote the above extractor procedure. In particular, for any fixed graph G, algorithm A, ROM H, input χ, and random coins R, $E(G, A, H, \chi, \sigma_i \| h_i)$ takes as input a state of the algorithm σ_i and the corresponding small additional hint h_i (as described above), and outputs a set of $|P'_i|$ input-output pairs of a ROM without ever querying them.

Theorem 5. *As long as* $\textsc{Collision}(A, H, \chi, R)$, $\textsc{BadOrder}(A, H, \chi, R)$, *and* $\textsc{MisColor}(G, A, H, \chi, R, \tau)$ *do not occur, running* $E^H(G, A, \chi, \sigma_i \| h_i)$ *will successfully extract* $|P'_i|$ *pairs* $(a_1, b_1), \ldots, (a_{|P'_i|}, b_{|P'_i|})$ *such that for all* $j \leq |P'_i|$ *(1)* $H(a_j) = b_j$ *and (2)* E *never queries* $H(a_j)$.

Proof. Recall that since $\textsc{Collision}(A, H, \chi, R)$, $\textsc{BadOrder}$, $\textsc{MisColor}$ do not occur, we are able to extract a unique Ex-Post-Facto Labeling $\ell'_1, \ldots, \ell'_n = \texttt{EPFL}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$, a unique Ex-Post-Facto Graph $G' = \texttt{EPFG}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$, and a unique legal Ex-Post-Facto Pebbling $P' = \texttt{EPFP}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$ of G' using definitions 8, 9, and 11. For each node $v \in P'_i$ let ℓ'_v be the label given by $\texttt{EPFL}(G, \texttt{trace}(A, H, \chi, R), \chi, \tau)$. By definition of P', $\texttt{need}(v, i) = 1$. Thus, while E^H is simulating the execution $A^H(\chi; R)$ from the starting state $\sigma_i \| h_i$, the label ℓ'_v is next used as input to a random oracle query *before* the query $\texttt{prelabel}(v, \vec{\ell'}, G, \chi)$ is made, i.e. there exists a query $x_{k,l} = w_1 \| \ell'_v \| w_2$, for some strings $w_1, w_2 \in \{0,1\}^*$, with $k > i$ identified by h^b_v. In particular, such a query must come before any query $x_{k,l}$ with $k \geq i$ that outputs ℓ'_v — the first of which is identified by h^a_v. Thus, when we encounter the query identified by h^a_j, we must have already encountered the query identified by h^b_j, and E^H can, as described, substitute ℓ'_v as the output for this query without actually querying $H(x_{k,l})$. Furthermore, any time the query $x_{k,l}$ would be made by $A^H(\chi; R)$ in the future, E^H can directly respond with the label ℓ'_v without querying $H(\ell'_v)$. Since the event $\textsc{MisColor}(G, A, H, \chi, R, \tau)$ does not occur, and any $v \in P'_i$ must be green, there must be some query $x_{k,l} = \texttt{prelabel}(v, \vec{\ell'}, G, \chi)$ in $\texttt{trace}(A, H, \chi, R)$ — the first such query will be identified by h^a_v. Since $\textsc{Collision}(A, H, \chi, R)$ does not occur, this is the only random oracle query A^H makes that produces ℓ'_v as output. This ensures that E^H can extract the pair $(\texttt{prelabel}(v, \vec{\ell'}, G, \chi), \ell'_v)$ for all $v \in P'_i$ without querying $\texttt{prelabel}(v, \vec{\ell'}, G, \chi)$. Note that since $\texttt{prelabel}(v, \vec{\ell'}, G, \chi)$ includes v iteself, we know all prelabels are distinct and thus each pair $(\texttt{prelabel}(v, \vec{\ell'}, G, \chi), \ell'_v)$ is distinct.

Hint Size We must consider the required size of h_i for any round i. First, we observe that h_i contains 3 elements for each node $v_j \in |P'_i|$. Let q be the total number of queries made by A and n be the number of bits in an input to H. h^b_j uniquely identifies a query made by A, and thus requires $\log_2(q)$ bits. h^w_j identifies a parent of each vertex, and thus requires at most $\log_2(\texttt{indeg}(G))$ bits.

h_j^a once again identifies a query and requires $\log_2(q)$ bits. Thus, for each node $v_j \in |P_i'|$, h_i contains at most $\log_2\left(q^2\mathsf{indeg}(G)\right)$ bits. We can upper bound this value as at most $2\log_2(q)+\log_2(\mathsf{indeg}(G))$ bits. The total length of h_i is therefore $(2\log_2(q)+\log_2(\mathsf{indeg}(G)))\,|P_i'|$.

We will now consider when an extractor is able to succeed at the Oracle Prediction Game. There are two necessary conditions for this: the extractor must be able to extract the $|P_i'|$ input-output pairs, and the hint given to the extractor must be sufficiently small. We define this with the event IE, for an *Implausible Extraction*.

Definition 13. *Fix a graph G, pROM algorithm A, input string χ and random coins R. Given a random oracle H and an integer i denoting an execution round we define the event* $\mathrm{IE}_i(G, A, H, \chi, R)$ *(implausible extraction during round i) as the event that* $\textsc{Collision}(A, H, \chi, R)$, $\textsc{BadOrder}(A, H, \chi, R)$, *and* $\textsc{MisColor}(G, A, H, \chi, R)$ do not *occur, but* $|\sigma_i| < \frac{|P_i'|\lambda}{4}$. *We further define the event* $\mathrm{IE}(G, A, H, \chi, R) = \bigcup_i \mathrm{IE}_i(G, A, H, \chi, R)$ *as the event that there exists some round where* $|\sigma_i| < \frac{|P_i'|\lambda}{4}$.

Lemma 3. *Given a graph G, pROM algorithm A, an input string χ, random coins R, a merkle tree root τ which is output by A, supposing that* $\lambda/4 \geq 2\log_2(q) + \log_2(\mathsf{indeg}(G'))$, $\Pr[\mathrm{IE}] \leq t2^{-\lambda/2}$, *where the probability is taken over the choice of the random oracle H and t upper bounds the number of execution rounds of our algorithm A.*

Proof. For $\mathrm{IE}_i(G, H, \chi, R, \tau)$ to occur, $\textsc{Collision}(A, H, \chi, R)$, $\textsc{BadOrder}(A, H, \chi, R)$, and $\textsc{MisColor}(G, A, H, \chi, R, \tau)$ must not occur. Thus, by theorem 5, the extractor is guaranteed to successfully output $|P_i'|$ input-output pairs (a_j, b_j) $(j \leq |P_i'|)$ for the random oracle H without every querying $H(a_j)$ for any j. Yet given the hint size we can upper bound such a probability using Lemma 1 (over the choice of H).

$$\begin{aligned}\Pr[\mathrm{IE}_i] &\leq 2^{(|\sigma_i|+|h_i|)}2^{-\lambda|P_i'|} \leq 2^{(\frac{|P_i'|\lambda}{4}+(2\log_2(q)+\log_2(\mathsf{indeg}(G')))|P_i'|)}2^{-\lambda|P_i'|} \\ &\leq 2^{(2\frac{|P_i'|\lambda}{4})}2^{-\lambda|P_i'|} \leq 2^{\frac{|P_i'|\lambda}{2}-\lambda|P_i'|} \leq 2^{(-\frac{|P_i'|\lambda}{2})}\ .\end{aligned}$$

We can see that the worst case here is $|P_i'| = 1$, and thus we can upper bound $\Pr[\mathrm{IE}] \leq \sum_{i=1}^{t}\Pr[\mathrm{IE}_i] \leq t2^{-\lambda/2}$.

Fix a graph G, pROM algorithm A, an input string χ, random coins R, and random oracle H. Let $tr = \mathtt{trace}(A, H, \chi, R)$ be the execution trace when running A on χ and τ be the Merkle Tree root output by A. As long as none of the bad events $\textsc{Collision}(A, H, \chi, R)$, $\textsc{BadOrder}(A, H, \chi, R)$, $\textsc{MisColor}(G, A, H, \chi, R, \tau)$, nor $\mathrm{IE}(G, A, H, \chi, R)$ occur, clearly the cumulative memory cost (cmc) of running A on χ is at least $\frac{\lambda}{4}cc(G')$, where $cc(G')$ is the cumulative pebbling cost of the Ex-Post-Facto Graph $G' = \mathtt{EPFG}(G, \mathtt{trace}(A, H, \chi, R), \chi, \tau)$. Thus, with probability $1-(\Pr[\textsc{Collision}]+\Pr[\textsc{BadOrder}]+\Pr[\textsc{MisColor}]+\Pr[\mathrm{IE}])$, the cmc is large as long as $\mathtt{cc}(G')$ is large.

Theorem 6. *Fixing a graph G, pROM algorithm A, an input string χ, and random coins R. Then, if the bad events* Collision, BadOrder, MisColor, *and* IE *do not occur,* $\mathsf{cmc}(tr) \geq \mathsf{cc}(G')\frac{\lambda}{4}$, *where* $tr = \mathtt{trace}(A, H, \chi, R)$ *is the execution trace when running A on χ,* $G' = \mathtt{EPFG}(G, \mathtt{trace}(A, H, \chi, R), \chi, \tau)$ *is the Ex-Post-Facto Graph, and τ is the Merkle Tree root output by A.*

Proof. Assuming that none of the bad events events occur, we have

$$\mathsf{cmc}(\mathtt{trace}) = \sum_i |\sigma_i| \geq \sum_i |P'_i|\lambda/4 = \mathsf{cc}(P')\lambda/4 \geq \mathsf{cc}(G')\lambda/4 \ .$$

The first inequality follows since $|\sigma_i| \geq |P'_i|\lambda/4$ by definition of the event IE (since Collision, BadOrder, MisColor do not occur). The next equality follows from the definition of cc and the final inequality follows since, by Theorem 4, P' is a legal pebbling of G' and therefore $cc(P') \geq cc(G')$. □

By applying the probability upper bounds of the bad events, we can also make this statement concrete.

5 Lucky Certificates

Definition 14. *Fix a graph G with N nodes, input string χ, random coins R, and algorithm A, a number of challenge rounds k, and a threshold β (with $0 < \beta$). Given some query $x_{i,j}$ made during round i, we define the event* $\textsc{LuckyQuery}_{i,j}(A, H, \chi, R, k, \beta)$ *as the event that the query $x_{i,j}$ is fresh and produces output string $\tau_{i,j} = H(x_{i,j})$ such that: (1) the Ex-Post-Facto Graph[5] $G'_{i,j} = \mathtt{EPFG}(G, \mathtt{trace}(A, H, \chi, R), \chi, \tau_{i,j})$ has at most $(1-\beta)N$ nodes - i.e. the fraction of red nodes is at least β - and (2) for each index $z \in [1, k]$, the challenge $c_z = (H(\chi, z, \tau_{i,j}) \bmod N)$ is a* Green *node in G - i.e. for the labeling $\ell'_1, \ldots, \ell'_n = \mathtt{EPFL}(G, \mathtt{trace}(A, H, \chi, R), \chi, \tau_{i,j})$ ℓ'_{c_z} is* Green*, i.e. that the query $x_{i,j}$ forms an accepted Merkle tree root. Intuitively, if the Ex-Post-Facto Graph $G'_{i,j}$ has at most $(1-\beta)N$ nodes then there are* at least *βN missing nodes and the probability that all of k of our challenges are green nodes is at most $(1-\beta)^k$. The event* $\textsc{LuckyQuery}_{i,j}$ *happens if all of the challenges are green even though a substantial fraction β of the nodes are red.*

Lemma 4. *For any graph G with N nodes, input string χ, random coins R, and algorithm A, number of challenges rounds k, and luck threshold β, let q be*

[5] An astute reader may notice that there is a subtle difference between calculating the Ex-Post-Facto Graph using the entire execution trace $\mathtt{trace}(A, H, \chi, R)$ versus calculating the Ex-Post-Facto Graph using the partial execution trace at the time that the query $x_{i,j}$ is made. However, as long as neither bad event $\textsc{Collision}(A, H, \chi, R)$ nor $\textsc{BadOrder}(A, H, \chi, R)$ occurs the Ex-Post-Facto Graph $\mathtt{EPFG}(G, \mathtt{trace}(A, H, \chi, R), \chi, \tau_{i,j})$ will be identical to the Ex-Post-Facto graph that we would have obtained by considering the partial execution trace.

the number of queries made by A. Then $\Pr_H[\textsc{LuckyQuery}(A,H,\chi,R,k,\beta) \mid \overline{\textsc{Collision}} \wedge \overline{\textsc{BadOrder}}] \leq q(1-\beta)^k$*, where the probability is taken over the choice of random oracle H and we use the abbreviations* $\textsc{Collision} = \textsc{Collision}(A,H,\chi,R)$ *and* $\textsc{BadOrder} = \textsc{BadOrder}(A,H,\chi,R)$*.*

Proof. First, we will show that for any round of compuation i and query within that round j, $\Pr_H[\textsc{LuckyQuery}_{i,j}(A,H,\chi,R,k,\beta) \mid \overline{\textsc{Collision}(A,H,\chi,R)} \wedge \overline{\textsc{BadOrder}(A,H,\chi,R)}] \leq (1-\beta)^k$. Consider the moment that the query $x_{i,j}$ is made by A. For $\textsc{LuckyQuery}_{i,j}$ to occur, this query must be fresh. Thus, since by definition Collision does not occur, the string $\tau_{i,j} = H(x_{i,j})$ is output for the first time. Since by definition BadOrder does not occur, this means no query which contains $\tau_{i,j}$ as a substring of the input (including, notably, all the challenge queries) has been made prior. Thus, for each index $z \in [1,k]$, the challenge $c_z = (H(\chi,z,\tau_{i,j}) \mod N$ can be viewed an an uniformly random integer in $\{0,\ldots,N-1\}$ which is selected independently of the other challenges. Let $R_{i,j} \subseteq \{0,\ldots,N-1\}$ denote the set of red-nodes. The probability that none of the k challenges lie in the set $R_{i,j}$ is at most $\left(\frac{N-|R_{i,j}|}{N}\right)^k$. If $|R_{i,j}| < \beta N$ then by definition the event $\textsc{LuckyQuery}_{i,j}$ does not occur. Otherwise, the probability that none of the k challenges lie in the set $R_{i,j}$ is at most $(1-\beta)^k$. It follows that $\Pr_H[\textsc{LuckyQuery}_{i,j}(A,H,\chi,R,k,\beta) | \overline{\textsc{Collision}(A,H,\chi,R)} \wedge \overline{\textsc{BadOrder}(A,H,\chi,R)}] \leq (1-\beta)^k$. The result follows by union bounding over all q queries. □

Theorem 7. *Fix a graph G with N nodes, input string χ, random coins R, and algorithm A, a number of challenges rounds k, and a threshold β (with* $0 < \beta$*). Given an algorithm A, if* $\textsc{Success}(G,H,\chi,R,A,k)$ *occurs and the bad events* Collision, BadOrder, MisColor, IE, *and* LuckyQuery *do not occur, then* $\mathsf{cmc}(tr) \geq \frac{\lambda}{4}\left(\min_{S \subset V(G): |S| \leq \beta N}(cc(G-S))\right)$*, where* $tr = \texttt{trace}(A,H,\chi,R)$ *is the execution trace when running A on χ.*

Proof. Since the algorithm, random oracle, input string, and random coins are given, we have a unique and well-defined $\texttt{trace}(A,H,\chi,R)$. A unique and well-defined database $D(\texttt{trace}(A,H,\chi,R))$, ex-post-facto labeling $\ell'_1,\ldots.\ell'_N = \texttt{EPFL}(G,\texttt{trace}(A,H,\chi,R),\chi,\tau)$, and Ex-Post-Facto Graph $G' = \texttt{EPFG}(G,\texttt{trace}(A,H,\in,R),\chi,\tau)$ follow by definitions 6, 8, and 9. Now, we can note that since $\textsc{Success}(G,H,\chi,R,A,k)$ occurred, no challenges to *Red* nodes may have been made, as such a challenge would, by definition, not have been verified. Further, since we know $\textsc{LuckyQuery}(A,H,\chi,R,k,\beta)$ does not occur, yet we have a root τ, output by A which passes condition (2) of definition 14, namely producing only *Green* challenges, it must be that the graph $G' = \texttt{EPFG}(G,\texttt{trace}(A,H,\chi,R),\chi,\tau)$ has more than $(1-\beta)N$ nodes - intuitively, we know that the Ex-Post-Facto Graph is suitably large. Therefore, since G' is a subset of G with fewer than βN nodes removed, we know $cc(G') > \left(\min_{S \subset V(G) || S| \leq \beta N}(cc(G-S))\right)$. By theorem 5, we know that since the bad events Collision, BadOrder, MisColor, and IE do not occur, the $\mathsf{cmc}(tr)$ is

lower bounded by $\mathsf{cc}(G')\frac{\lambda}{4}$. It follows that $\mathsf{cmc}(tr) \geq \frac{\lambda}{4}\left(\min_{S\subset V(G)||S|\leq\beta N}(cc(G-S))\right)$. □

Thus far, we have been writing theorems which are conditional that bad events do not occur. Now, we instead account for the probability of bad events.

Corollary 1. *Fix a graph G with N nodes and maximum in-degree δ, input string χ, random coins R, and algorithm A, a number of challenges rounds k, and a threshold β (with $0<\beta$). Given an algorithm A which succeeds with probability at least α after t rounds of execution, $\Pr_H[\mathsf{cmc}(tr) < \frac{\lambda}{4}\left(\min_{S\subset V(G)||S|\leq\beta N}(cc(G-S))\right)] \leq \alpha - \left(2^{-\lambda}\binom{q}{2} + n2^{-\lambda}q^2 + 2^{-\lambda}|V(G)| + 2^{-\lambda/2}t\ + q(1-\beta)^k\right)$, where $tr = \mathtt{trace}(A,H,\chi,R)$ is the execution trace when running A on χ and probability is taken over the choice of random oracle.*

Proof. If an algorithm succeeds with probability α, then it succeeds and the bad events Collision, BadOrder, MisColor, IE, and LuckyQuery do not occur with probability *at least* $\alpha - (\Pr[\textsc{Collision}] + \Pr[\textsc{BadOrder}] + \Pr[\textsc{MisColor}] + \Pr[\mathrm{IE}] + \Pr[\textsc{LuckyQuery}])$. By lemmas 7, 10, 2, 3, and4 , this probability is thus at most $\alpha - \left(2^{-\lambda}\binom{q}{2} + n2^{-\lambda}q^2 + 2^{-\lambda}|V(G)| + 2^{-\lambda/2}t + q(1-\beta)^k\right)$, where q upper bounds the number of queries, and t the number of execution rounds, of our algorithm A. By Theorem 7, if these events do not occur then the cmc is lower bounded. □

6 Instantiations of Merkle Tree Memory-Hard Proofs of Work

In the previous section, we proved that the cmc of any attacker which produces a valid Memory-Hard Merkle Tree with high probability proof is at least $\frac{\lambda}{4}\left(\min_{S\subset V(G)||S|\leq\beta N}(cc(G-S))\right)$. The question remaining is: can we find graphs G such that cc is large even when S is relatively large?

We say that a DAG $G=(V,E)$ is (e,d)-depth-robust if for all subsets $S\subseteq V$ the graph $G-S$ has a directed path of length at least d. Alwen et al. [ABP17] showed that $\mathsf{cc}(G)\geq ed$ for any (e,d)-depth-robust graph. One can trivially observe that if $G=(V,E)$ is (e,d)-depth-robust and we remove $|S|<e$ nodes that the remaining graph $G'=G-S$ is still $(e-|S|,d)$-depth-robust. It immediately follows from [ABP17] that $\mathsf{cc}(G')\geq(e-|S|)d$. Generalizing this observation,

Lemma 5. *Let $e'<e$ be given. Then for any (e,d)- depth-robust graph G we have:*

$$\min_{S\subseteq V:|S|\leq e'} \mathsf{cc}(G-S) \geq (e-e')d\ .$$

In our setting we will think of S as denoted the set of red-nodes that were erased from the original graph. If we assume that the event $\textsc{LuckyQuery}(A,H,\chi,R,k,\beta)$ does not occur then $|S|\leq\beta N$. It follows that our Ex-Post-Facto Graph G' has $\mathsf{cc}(G')\geq\min_{S\subseteq V:|S|\leq\beta N}\mathsf{cc}(G-S)\geq(e-\beta N)d$. As long as we pick β such that $\beta N\leq e/2$ we have $\mathsf{cc}(G')\geq ed/2$.

Alwen et al.[ABH17,ABP17] constructed (e, d)-depth robust graphs G with constant indegree, $e = \Omega(N/\log N)$ and $d = \Omega(N)$. If we allow the indegree of the graph to be $O(\log N)$ then for any constant $\epsilon > 0$, Alwen et al. [ABP18] constructed ϵ-extreme depth-robust graphs — graphs that are (e, d)-depth robust for *any* parameters e, d such that $e + d \leq (1 - \epsilon)N$. However, in practical applications we generally want constant indegree.

Finally, we instantiate our Memory-Hard Proof of Work with the family of graphs $\{G_N = ([N], E_N)\}_{N=1}^{\infty}$ such that each graph G_N is $(c_3 N/\log N, c_4 N)$ depth-robust for fixed constants $c_3, c_4 > 0$. In our analysis we will set $\beta = c_3/(2\log N)$ so that $\beta N \leq c_3 N/(2\log N)$ and that for any $|S| \leq \beta N$ we have $\mathsf{cc}(G - S) \geq c_3 c_4 N^2/(2\log N)$ i.e., the graph $G - S$ still has high cumulative pebbling cost. We instantiate the parameter k (number of challenges) as $k = \lambda(2\log N)/C_3$ to ensure that the error term $q(1-\beta)^k \leq qe^{-\lambda}$ remains negligible.

Corollary 2. *Let G be any (e, d)-depth-robust graph with N nodes and set $k = \left(\frac{2N\ln 2}{e}\lambda\right)$. For any input string χ and any algorithm A and any string R (A's random coins). Let $\alpha_N \doteq \Pr[\textsc{Success}(G, H, \chi, R, A, k)]$ denote the probability that A succeeds taken over the selection of the random oracle H. We have $\Pr_H[\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) < \frac{\lambda}{4}(ed/2)] \leq \alpha_N - f(N, q, \lambda, \alpha_N)$ where the probability is taken over the selection of H and $f(N, q, \lambda, \alpha_N) = \left(2^{-\lambda}\binom{q}{2} + n2^{-\lambda}q^2 + 2^{-\lambda}N + 2^{-\lambda/2}t + q2^{-\lambda}\right)$ is a negligible function.*

Proof. In our analysis we set $\beta = \frac{e}{2N}$ so that $\beta N = e/2$. Applying Lemma 5 with $e' = e/2$ we have $\min_{S\subseteq V:|S|\leq e/2} \mathsf{cc}(G - S) \geq ed/2$. Since $k = \left(\frac{2N\ln 2}{e}\lambda\right)$, our upper bound for the event $\textsc{LuckyQuery}(A, H, \chi, R, k, \beta)$ is $q(1-\beta)^k \leq q(1 - \frac{e}{2N})^{\frac{2N}{e}\lambda\ln 2} \leq q2^{-\lambda}$.

Therefore, by Corollary 1, we have $\Pr_H[\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) < \frac{\lambda}{4}(ed/2)] \leq f(N, q, \lambda, \alpha_N)$ where $f(N, q, \lambda, \alpha_N) = \alpha_N - \left(2^{-\lambda}\binom{q}{2} + n2^{-\lambda}q^2 + 2^{-\lambda}N + 2^{-\lambda/2}t + qe^{-\lambda}\right)$ □

6.1 Instantiation with DRSample

Fact 8 *There exist constants $c_1, c_2 > 0$ and a family of graphs $\{G_N = ([N], E_N)\}_{N=1}^{\infty}$ such that for each N we have $\mathsf{indeg}(G_N) = 2$ and G_N is $(c_1 N/\log N, c_2 N)$-depth-robust. In particular, for each N we have*

$$\min_{S\subseteq V:|S|\leq c_1 N/(2\log N)} \mathsf{cc}(G - S) = \frac{(c_1 c_2/2)N^2}{\log N} \ .$$

Proof. By Alwen et al.[ABH17,ABP17] there exist constants c_1, c_2 such that there is a family of graphs $\{G_N = ([N], E_N)\}_{N=1}^{\infty}$ such that for each N we have $\mathsf{indeg}(G_N) = 2$ and such that G_N is $(c_3 N/\log N, c_4 N)$-depth-robust. The Theorem now follows by setting $c_1 = c_3/2$ and $c_2 = c_3 c_4/2$ and applying Lemma 5 with $e' = (c_3/2)N/\log N$. In particular, we have $\min_{S\subseteq V:|S|\leq e'} \mathsf{cc}(G - S) \geq (e - e')d \geq (e/2)d \geq (c_3 c_4/2)N^2/\log N$. □

The proof of Corollary 3 follows immediately from Corollary 2. Observe that since $k = (2\lambda \log N)/c_1$ and each Merkle Tree opening is $O(\lambda \log N)$ bits the overall proof size is $O(\lambda^2 \log^2 N)$ bits which is sublinear with respect to N.

Corollary 3. *Let $\{G_N = ([N], E_N)\}_{N=1}^{\infty}$ be the family of graphs from Fact 8. Let $c_1, c_2 > 0$ be the constants such that G_N is $(c_1 N/\log N, c_2 N)$-depth-robust and set $k = (2\lambda \log N)/c_1$. Fix any input string χ. Let A be any algorithm, let R be any string (random coins) and let $\alpha_N \doteq \Pr[\textsc{Success}(G_N, H, \chi, R, A, k)]$ denote the probability that A succeeds where the randomness is taken over the selection of the random oracle H. We have*

$\Pr_H[\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) < \frac{\lambda}{4}(c_3 c_4/2)N^2/\log N] \leq f(N, q, \lambda, \alpha_N)$ *where* $f(N, q, \lambda, \alpha_N) = \alpha_N - \left(2^{-\lambda}\binom{q}{2} + n2^{-\lambda}q^2 + 2^{-\lambda}N + 2^{-\lambda/2}t + qe^{-\lambda}\right)$

We also note that since $k = (2\lambda \log N)/c_3$, the proof size is sublinear with respect to N.

6.2 Instantiation with Argon 2i

We can also consider instantiating this with other families of graphs, such as those used in Argon2i. Argon 2i is (e, d)-depth-robust with $d = \tilde{\Omega}(N^3/e^3)$ for any $e \geq n^{2/3}$. If we set $k = \Omega(N\lambda/e)$, it follows from Corollary 2 that we obtain a MH Proof of work with cmc at least $\tilde{\Omega}\left(N^3/e^2\right)$

Corollary 4. *Let $\{G_N = ([N], E_N)\}_{N=1}^{\infty}$ be the family of graphs used in Argon2i. Let $c_1, c_2 > 0$ be the constants such that G_N is $(c_1 N/\log N, c_2 N)$-depth-robust and set $k = (2\lambda \log N)/c_1$. Fix any input string χ. Let A be any algorithm, let R be any string (random coins) and let $\alpha_N \doteq \Pr[\textsc{Success}(G_N, H, \chi, R, A, k)]$ denote the probability that A succeeds where the randomness is taken over the selection of the random oracle H. We have*

$\Pr_H[\mathsf{cmc}(\mathtt{trace}(A, H, \chi, R)) <$
$\frac{\lambda}{4}\tilde{\Omega}\left(N^3/e^2\right)] \leq f(N, q, \lambda, \alpha_N)$ *where* $f(N, q, \lambda, \alpha_N) = \alpha_N - \left(2^{-\lambda}\binom{q}{2} + n2^{-\lambda}q^2 + 2^{-\lambda}N + 2^{-\lambda/2}t + qe^{-\lambda}\right)$

This means there is a trade-off when selecting our e: the greater the cmc achieved, the larger the proof size. For example, we achieve a maximal cmc lower bound by setting $e = n^{2/3}$ so that cmc is $\tilde{\Omega}\left(N^{5/3}\lambda\right)$, but the proof size $k = \Omega(N^{1/3}\lambda)$ is larger. It possible to reduce the proof size k by increasing e, but this will reduce CMC e.g., if we set $N = N^{0.9}$ then we have $k = \Omega(N^{0.1}\lambda)$ but the CMC lower bound drops to $\Omega(N^{1.2})$.

7 General Attacks on Merkle Tree Memory Hard Proofs of Work

In this section we outline general attacks on Memory Hard Proofs of Work which follow the Merkle Tree Proofs framework outlined earlier in the paper. We show that any scheme which follows the Merkle Tree Proofs framework and uses a

graph with constant indegree can be meaningfully attacked unless $k \in \omega(\log N)$ and that a scheme which uses Argon2 can be meaningfully attacked unless $j \in \Omega(N^{5/12})$.

7.1 Attack 1

Suppose that the underlying DAG G is (e, d)-reducible meaning that there is a set $S \subseteq V(G)$ of size $|V(S)| = e$ such that $G - S$ contains no directed path of length d. The attack is to set $\ell'_v = 0^\lambda$ for all $v' \in S$ and then compute the remaining nodes in parallel following the honest labeling rule i.e., for a node $v \notin S$ with $\mathsf{parents}(v, G) = v_1, \ldots, v_\delta$ we set $\ell'_v = H(\chi, v, \ell'_{v_1}, \ldots, \ell'_{v_k})$. Finally, the attack will compute the Merkle-Tree root τ using the labeling ℓ', compute the challenges $c_i = (H(\chi, i, \tau) \mod N$. Note that because we follow the honest labeling rule for all nodes $v \notin S$ that there are at most e red nodes so with probability at least $(1 - e/N)^k$ it will be possible to respond to all challenges. Because the graph $G-S$ has no directed path of length d it is possible to compute the labels $\ell'_0, \ldots, \ell'_{N-1}$ in parallel time *at most* d and it takes at most $O(\log N)$ rounds to compute the Merkle-Tree root and respond to any challenges. Thus, the attack has cumulative memory cost *at most* $O(Nd\lambda + N\lambda \log N)$.

The above attack is reminiscent of the parallel pebbling attacks of Alwen and Blocki [AB16]. The primary difference is that we can *start* the pebbling attack with pebbles on all nodes in our depth-reducing set allowing us to achieve lower costs. However, the attack is only effective if the probability of success $(1 - e/N)^k$ is sufficiently large. We now discuss the implications of our attack on MT based Memory Hard Proofs of Work. In particular, we can user our attack to *lower bound* the number

General Attack on any DAG G It is known that any DAG G with constant indegree is (e_i, d_i)-reducible with $e_i = O(Ni/\log N)$ and $d_i = N/2^i$ for any parameter $1 \leq i \leq \log_2 N$ [AB16]. Thus, for any $i \leq \log_2 N$ we obtain an attack with cumulative memory cost *at most* $O(N^2/2^i\lambda + N\lambda \log N)$ and which succeeds with probability at least $(1 - i/\log N)^{k_i} \approx (1/e)^{k_i}$ where we define $k_i = \frac{ki}{\log N}$. In particular, if $k = O((\log N)/i)$ then $k_i = O(1)$ and the attack will succeed with constant probability. As a concrete example, suppose that $k = O(\log N/\log\log N)$. In this case we can set $i = 100 \log\log N$ and the attack will succeed with constant probability and the cost will be at most $O(N^2\lambda/log^{100} N + N\lambda \log N)$ which is significantly lower than $O(N^2\lambda/\log N)$! Thus, for any MT based Memory Hard Proof of Work based on any DAG G we must set $k = \omega(\log N/\log\log N)$ to achieve strong memory-hardness. Recall that in our construction we set $k = O(2\lambda \log N)$. The above analysis indicates that the dependence on $\log N$ cannot be eliminated.

Attack on Argon2i-MTP We can further tighten this bound with specific families of graphs. An Argon2i DAG G is (e_i, d_i)-reducible with $d_i = O(N^3/e_i^3)$ for any parameter i, as shown by [BZ17]. Applying attack 1 this observation leads to an attack with cumulative memory cost $O(N^4\lambda/e_i^3 + N\lambda \log N)$ that succeeds

with probability $(1-e_i/N)^k$. Setting $e_i = N/10k$ we have $(1-e_i/N)^k \geq 0.5$ and cumulative memory cost $O(Nk^3\lambda + N\lambda \log N)$. For example, if we set $k = N^{0.1}$ then there is a attack with cumulative memory cost at most $O(N^{1.3}\lambda)$ which succeeds with constant probability. To completely prevent attack 1 it is necessary to set $k = N^{1/3}$. However, even in this case one can still use the general pebbling attacks of [AABN02,BZ17] to solve the POW with cost at most $O(\lambda N^{1.768})$.

7.2 Attack 2

We can adopt the recursive pebbling strategy of [ABP17] to obtain an even more efficient attack when the graph is (e_i, d_i)-reducible on a range of points (e_i, d_i). We start by "cheating" and placing pebbles on all nodes in S_0 where $|S_0| \leq e_0$ is a (e_0, d_0)-reducing set. We now want to pebble the remaining graph $G - S_0$ in parallel. Attack 1 simply exploits the fact that $G - S_0$ contains no directed path of length d_0 to pebble the graph in time at most d_0. However, this pebbling strategy may require up to N pebbles each round bringing the total cumulative pebbling cost to $\Omega(Nd_0)$. Following the strategy of [ABP17] we can partition the nodes of $G - S_0$ into $2d_0$ sets $D_1, \ldots, D_{2d_0}$ such that 1) $|D_i| \leq N/d_0$ for every $i \leq 2d_0$, and 2) for every $i < j$ there is no directed edge from any node in D_j to any node in D_i.

Lemma 6. *[ABP17, Lemma 8] Let $G = (V, E)$ be any DAG of size N, indegree δ, and $\mathsf{depth}(G) \leq d_0$. If G is (e_1, d_1) depth-reducible with parameters e_1, d_1 such that $2d_1N \leq e_1d_0$ and $d_1 \leq d_0$, then for any target set $T \subseteq V$ we have*
$\mathsf{cc}(G, T, 2d_0) \leq (4\delta+4)e_1d_0 + \frac{N}{e_1}\left(\max_{T' \subseteq V - S_1; |T'| \leq \delta \cdot e_1} \mathsf{cc}(G - S_1, T', 2d_1)\right)$,
where $S_1 \subseteq V$ and $|S_1| \leq e_1$ such that $\mathsf{depth}(G - S_1) \leq d_1$

Theorem 9. *Let G be a directed acyclic graph with $N = |V|$ nodes and maximum indegree δ. Suppose that for some constant $\alpha > 0$ the graph G is (e, d)-reducible with $d = \alpha N^3/e^3$ for any $e \geq 1$. Then there is a constant $c > 0$ such that for every $k \geq 1$ there is a set $S_0 \subseteq V$ of size $|S_0| \leq N/k$ such that $\mathsf{cc}(G - S_0) \leq cNk^{12/5} = O\left(Nk^{12/5}\right)$*

We can apply this attack to Argon2i graphs to further refine our bounds. Recall that, with high probability, an Argon2i DAG G is (e_i, d_i)-reducible with $d_i = O(N^3/e_i^3)$ for any parameter i, as shown by [BZ17]. Thus, we can apply Theorem 9 to obtain an attack on Argon2i graphs with cumulative memory cost at most $O(Nk^{12/5})$, since we can (likely) delete $O(N/k)$ nodes from G without getting caught within k challenges. If $k \in o(N^{.320})$ then we have an attack with $cc \in o(N^{1.768})$ which is better than any known pebbling attack on Argon2i without node deletions [BZ17]. Similarly, if $k \in O(N^{5/18-\epsilon})$ then we have an attack with $cc \in O(N^{1.75-12\epsilon/5})$ which is provably better than *any* possible pebbling of G since [BZ17] showed that (whp) $\mathsf{cc}(G) = \tilde{\Omega}(N^{1.75})$. To make the attack completely ineffective one would need to use at least $k \in \Omega\left(N^{5/12}\right)$ challenges so that the attack has $\mathsf{cc} \in \Omega(N^2)$.

Theorem 10. *Let G be any directed acyclic graph with $N = |V|$ nodes and maximum indegree δ. There is a constant $c_\delta > 0$ such that for every $k \geq 1$ there is a set $S_0 \subseteq V$ of size $|S_0| \leq N/k$ such that:*

$$\begin{cases} \mathsf{cc}(G - S_0) \leq \frac{N^2}{(\log^8 N)} & \text{if } \; k < \frac{c_\delta \log N}{8 \log \log N} \\ \mathsf{cc}(G - S_0) \leq \frac{c_\delta N^2 \log \log N}{\left(2^{\frac{\log N}{c_\delta k}} \log N\right)} & \text{otherwise} \end{cases}$$

We can apply this theorem to obtain an attack attack on all graphs with constant indegree. Since we can, with constant probability, delete $O(N/k)$ nodes from G while still generating a certificate which passes k challenges, we know by theorem 10 that for any DAG with constant indegree there exists a set S_0, such that by "cheating" and placing pebbles on S_0,

$$\begin{cases} \mathsf{cc}(G - S_0) \leq \frac{N^2}{(\log^8 N)} & \text{if } \; k < \frac{c_\delta \log N}{8 \log \log N} \\ \mathsf{cc}(G - S_0) \leq \frac{c_\delta N^2 \log \log N}{\left(2^{\frac{\log N}{c_\delta k}} \log N\right)} & \text{otherwise} \end{cases} .$$

Suppose that we select a $k < \frac{c_\delta \log N}{8 \log \log N}$. Then we have shown an attack on any constant-indegree DAG with cmc at most $O(N^2/\log^8 N)$, which is superior to any possible pebbling of G. Suppose, on the other hand, that we select a $k \geq \frac{c_\delta \log N}{8 \log \log N}$. If $k \in o(\log N)$, then we have an attack with cumulative memory complexity $\frac{N^2 \log \log N}{2^{\omega(1)} \log N}$, which is better than the best known pebbling attack on G. Similarly, if $k \in o\left(\frac{\log N}{\log \log N}\right)$, we have an attack with cumulative memory complexity $\frac{N^2 \log \log N}{2^{\omega(\log \log N)} \log N}$, which is better than any possible pebbling of G. If $k \in o\left(\frac{\log N}{\log \log \log N}\right)$, then this attack has cmc at most $\frac{N \log \log N}{\omega(\log \log N) \log N}$, which is better than the pebbling lower bound of $\frac{N^2}{\log N}$.

Acknowledgments: This work was supported in part by NSF CAREER Award #2047272.

References

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 418–433, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer Berlin Heidelberg, Germany.

[AB16] Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 241–271, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[AB17] Joël Alwen and Jeremiah Blocki. Towards practical attacks on Argon2i and balloon hashing. In *2017 IEEE European Symposium on Security and Privacy*, pages 142–157, Paris, France, April 26–28, 2017. IEEE Computer Society Press.

[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 2017: 24th Conference on Computer and Communications Security*, pages 1001–1017, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[ABP17] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In Coron and Nielsen [CN17], pages 3–32.

[ABP18] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In Nielsen and Rijmen [NR18], pages 99–130.

[ACP+17] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In Coron and Nielsen [CN17], pages 33–62.

[AGK+18] Joël Alwen, Peter Gazi, Chethan Kamath, Karen Klein, Georg Osang, Krzysztof Pietrzak, Leonid Reyzin, Michal Rolinek, and Michal Rybár. On the memory-hardness of data-independent password-hashing functions. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *ASIACCS 18: 13th ACM Symposium on Information, Computer and Communications Security*, pages 51–65, Incheon, Republic of Korea, April 2–6, 2018. ACM Press.

[AGP24] Benedikt Auerbach, Christoph U. Günther, and Krzysztof Pietrzak. Trapdoor memory-hard functions. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part III*, volume 14653 of *Lecture Notes in Computer Science*, pages 315–344, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[AR17] Leo Alcock and Ling Ren. A note on the security of equihash. In *Proceedings of the 2017 on Cloud Computing Security Workshop*, CCSW '17, page 51–55, New York, NY, USA, 2017. Association for Computing Machinery.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 595–603, Portland, OR, USA, June 14–17, 2015. ACM Press.

[B+02] Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[BDK16] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy*, pages 292–302, Saarbrücken, Germany, March 21–24, 2016. IEEE Computer Society Press.

[BHK+19] Jeremiah Blocki, Benjamin Harsha, Siteng Kang, Seunghoon Lee, Lu Xing, and Samson Zhou. Data-independent memory hard functions: New attacks

and stronger constructions. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 573–607, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[BK15] Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. Cryptology ePrint Archive, Paper 2015/946, 2015.

[BK16] Alex Biryukov and Dmitry Khovratovich. Egalitarian computing. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016: 25th USENIX Security Symposium*, pages 315–326, Austin, TX, USA, August 10–12, 2016. USENIX Association.

[BL24] Jeremiah Blocki and Seunghoon Lee. A tight lower bound on the tdscrypt trapdoor memory-hard function. *Cryptology ePrint Archive*, 2024.

[BLZ21] Jeremiah Blocki, Seunghoon Lee, and Samson Zhou. On the security of proofs of sequential work in a post-quantum world. In Stefano Tessaro, editor, *ITC 2021: 2nd Conference on Information-Theoretic Cryptography*, volume 199 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:27, Seattle, WA, USA, July 23–26, 2021. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[BP17] Alex Biryukov and Léo Perrin. Symmetrically and asymmetrically hard cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 417–445, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.

[BZ17] Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of Argon2i. In Kalai and Reyzin [KR17], pages 445–465.

[CFHL21] Kai-Min Chung, Serge Fehr, Yu-Hsuan Huang, and Tai-Ning Liao. On the compressed-oracle technique, and post-quantum security of proofs of sequential work. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 598–629, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.

[CN17] Jean-Sébastien Coron and Jesper Buus Nielsen, editors. *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland.

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Nielsen and Rijmen [NR18], pages 451–467.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 585–605, Santa Barbara, CA, USA, August 16–20, 2015. Springer Berlin Heidelberg, Germany.

[DKW11] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 125–143, Providence, RI, USA, March 28–30, 2011. Springer Berlin Heidelberg, Germany.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*,

volume 740 of *Lecture Notes in Computer Science*, pages 139–147, Santa Barbara, CA, USA, August 16–20, 1993. Springer Berlin Heidelberg, Germany.

[DN17] Itai Dinur and Niv Nadler. Time-memory tradeoff attacks on the MTP proof-of-work scheme. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 375–403, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

[KR17] Yael Kalai and Leonid Reyzin, editors. *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 369–378, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013: 4th Innovations in Theoretical Computer Science*, pages 373–388, Berkeley, CA, USA, January 9–12, 2013. Association for Computing Machinery.

[NBF+16] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[NR18] Jesper Buus Nielsen and Vincent Rijmen, editors. *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[Pie19] Krzysztof Pietrzak. Proofs of catalytic space. In Avrim Blum, editor, *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 59:1–59:25, San Diego, CA, USA, January 10–12, 2019. Leibniz International Proceedings in Informatics (LIPIcs).

[RD17] Ling Ren and Srinivas Devadas. Bandwidth hard functions for ASIC resistance. In Kalai and Reyzin [KR17], pages 466–492.

A Proofs of Bad Event Lemmas

Lemma 7. *Let A be any pROM algorithm making at most q queries to the random oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$. For all inputs χ and random coins R we have $\Pr_H[\textsc{Collision}(A, H, \chi, R)] \leq \binom{q}{2} 2^{-\lambda}$ where the probability is taken over the selection of the random oracle H.*

Proof. Since for $\textsc{Collision}(A, H, \chi, R)$ to occur there must be some first collision, we examine only the first time that a collision occurs. Consider the set of queries A makes, Q and let $\textsc{Collision}_{i,j}$ be the event that query $x_{i,j}$ is fresh and there exists some prior query $x_{k,\ell} \neq x_{i,j}$ such that $k < i$ or both $k = i$ and $\ell < j$ but $H(x_{k,\ell}) = H(x_{i,j})$. Since $x_{i,j}$ is fresh, for any query $x \neq x_{i,j}$ we have $\Pr_H[H(x_{i,j}) = H(x)] \leq 2^{-\lambda}$. By union bounding over all queries $q \in Q_{\leq i}$ we

have $\Pr_H[\textsc{Collision}_{i,j}] \leq (|Q_{<i}| + j - 1)\, 2^{-\lambda}$. Now we have

$$\Pr_H[\textsc{Collision}(A, H, \chi, R)] \leq \sum_{i,j} \Pr_H[\textsc{Collision}_{i,j}] = 2^{-\lambda} \sum_{i=0}^{q-1} i = 2^{-\lambda} \binom{q}{2} .$$

□

Lemma 8. *Let A be any pROM algorithm making at most $q_{\leq i}$ queries to the random oracle $H : \{0,1\}^n \to \{0,1\}^\lambda$ in $Q_{\leq i}$, and j be an index such that $x_{i,j}$ is fresh. For all inputs χ and random coins R we have $\Pr_H[\textsc{BadOrder}_{i,j}(A, H, \chi, R)] \leq n2^{-\lambda} q_{\leq i}$ where the probability is taken over the selection of the random oracle H.*

Proof. Without loss of generality assume $x_{i,j}$ is fresh; otherwise we already have $\textsc{BadOrder}_{i,j}(A, H, \chi, R) = 0$. Thus, we may view $H(x_{i,j})$ as a uniformly random string. Thus, for any other λ-bit string s, $\Pr_H[H(x_{i,j}) = s] = 2^{-\lambda}$. For any string S, with n bits (where $n \geq \lambda$), there are $n - \lambda + 1$ substrings of length λ. Thus,

$$\begin{aligned} \Pr_H[H(q) \text{ is a substring of } S] &= \sum_{i=1}^{n-\lambda+1} \Pr_H[H(q) = S[i, i+\lambda-1]] \\ &= (n - \lambda + 1)2^{-\lambda} < n2^{-\lambda} . \end{aligned}$$

Consider all prior queries in $Q_{\leq i}$. Each query is a string of length n. Thus, since the probability $H(x_{i,j})$ is a substring of a given string of length n is $\leq n2^{-\lambda}$, and there are at most $q_{\leq i}$ prior input strings, by union bound we have

$$\begin{aligned} \Pr_H[\textsc{BadOrder}_{i,j}(A, H, x, R)] &\leq \sum_{k,\ell} \Pr_H[H(x_{i,j}) \text{ is a substring of } x_{k,\ell}] \\ &\leq \sum_{k=0}^{k=q_{\leq i}} n2^{-\lambda} \leq n2^{-\lambda} \sum_{k=0}^{k=q_{\leq i}} 1 \\ &\leq n2^{-\lambda} q_{\leq i} \end{aligned}$$

□

Lemma 9. *Let A be any pROM algorithm making at most $q_{\leq i}$ queries to the random oracle $H : \{0,1\}^n \to 2^\lambda$ in $Q_{\leq i}$, and q_i distinct fresh queries in Q_i. For all inputs χ and random coins R we have $\Pr_H[\textsc{BadOrder}_i(A, H, \chi, R)] \leq n2^{-\lambda} q_{\leq i} q_i$ where the probability is taken over the selection of the random oracle H.*

Proof. We know $\Pr_H[\textsc{BadOrder}_{i,j}(A, H, \chi, R)] \leq n2^{-\lambda} q_{\leq i}$ for any index j such that $x_{i,j}$ is fresh. Recall $\textsc{BadOrder}_i(A, H, \chi, R)$ is the union of $\textsc{BadOrder}_{i,j}(A, H, \chi, R)$

for all fresh queries. Since we know there are q_i fresh queries in Q_i, by union bound this means

$$\begin{aligned}\Pr_H[\textsc{BadOrder}_i(A,H,\chi,R)] &\leq \sum_{k=0}^{k=q_i} n2^{-\lambda}q_{\leq i} \\ &\leq n2^{-\lambda}q_{\leq i}\sum_{k=0}^{k=q_i} 1 \\ &\leq n2^{-\lambda}q_{\leq i}q_i\end{aligned}$$

□

Lemma 10. *Let A be any pROM algorithm making at most q queries over r rounds to the random oracle $H : \{0,1\}^n \to 2^\lambda$. For all inputs χ and random coins R we have*
$\Pr_H[\textsc{BadOrder}(A,H,\chi,R)] \leq q^2n2^{-\lambda}$ *where the probability is taken over the selection of the random oracle H.*

Proof. Recall that $\textsc{BadOrder}(A,H,\chi,R) = \bigcup_i \textsc{BadOrder}_i(A,H,\chi,R)$. Thus, by a simple union bound we have

$$\begin{aligned}\Pr_H[\textsc{BadOrder}(A,H,\chi,R)] &\leq \sum_{i=0}^{i=r}\Pr_H[\textsc{BadOrder}_i(A,H,\chi,R)] \\ &\leq \sum_{i=0}^{i=r} q_{\leq i}q_i n2^{-\lambda} \\ &\leq qn2^{-\lambda}\sum_{i=0}^{i=r} q_i \\ &= q^2n2^{-\lambda}\end{aligned}$$

□

B Proofs of Theorems 9 and 10

B.1 Proof of Theorem 9

First, a useful lemma.

Lemma 11. *Suppose that $G = (V,E)$ is a DAG with $N = |V|$ nodes, maximum indegree $\delta = 2$, maximum depth $\mathsf{depth}(G) \leq d_0$ and that for some constant $\alpha \geq 0$, the graph G is (e,d)-reducible with $d = \alpha\left(\frac{N}{e}\right)^3$ for any $e \geq 1$. Fix a sequence $B = b_1,\ldots$ such that $0 < b_1 \leq 1$ and $\forall j > 0$ $b_j \leq \frac{3}{4\log_k(2)}b_{j-1}$ and let $e_i = Nk^{-b_i}$ and $d_i = \alpha k^{3b_i}$. Similarly fix sets $S_1, S_2, \ldots$ such that $|S_i| \leq e_i$, $S_i \subseteq V(G) \setminus (S_1,\ldots,S_{i-1})$ and $\mathsf{depth}(G - \cup_{j=1}^{i} S_i) \leq d_i$. For any constant recursion depth i and any target set $T \subseteq V$ we have*

$\mathsf{cc}(G, T, 2d_0) = c_{i,\delta} N k^{3-b_1} + c_{i,\delta} \sum_{j=1}^{i} N k^{b_1+\ldots+b_{i-2}+4b_{i-1}-b_i}$
$+ c_{i,\delta} k^{b_1+\ldots+b_i} \left(\max_{T' \subseteq V \setminus S_i} \mathsf{cc}(G - S_i, T', 2d_i)\right)$, *where* $|S_i| \leq e_i$ *where* $c_{i,\delta} > 0$ *is a a constant that depends on* δ, α *and* i.

Proof. We prove this by induction.

First, we note that since $e_i = Nk^{-b_i}$ and $d_i = \alpha k^{3b_i}$ for each $i \geq 1$, for all i we have

$$b_i \leq \frac{3}{4\log_k(2)} b_{i-1} \tag{1}$$
$$\log_k(2) 4b_i \leq 3b_{i-1} \tag{2}$$
$$2k^{4b_i} \leq k^{3b_{i-1}} \tag{3}$$
$$2\alpha k^{3b_i} N \leq N k^{-b_i} \alpha k^{3b_{i-1}} \tag{4}$$
$$2d_i N \leq e_i d_{i-1} \tag{5}$$

We note this also implies $d_i \leq d_{i-1}$.

Since both these statements are true by our definition of the sequence B, we can invoke Lemma 6.

Our inductive base cases is $i = 1$. Let S_1 be given such that $|S_1| \leq e_1$ and $\mathsf{depth}(G - S_1) \leq d_1$. Applying Lemma 6 we have

$$\begin{aligned}
\mathsf{cc}(G, T, 2d_0) &\leq (4\delta + 4) e_1 d_0 + \frac{N}{e_1} \left(\max_{T' \subseteq V \setminus S_1} \mathsf{cc}(G, T', 2d_0) \right) \\
&\leq 12 N k^{-b_1} (\alpha k^{3b_0}) + k^{b_1} \left(\max_{T' \subseteq V \setminus S_1} \mathsf{cc}(G, T', 2d_0) \right) \\
&\leq 12 \alpha N k^{-b_1+3b_0} + k^{b_1} \left(\max_{T' \subseteq V \setminus S_1} \mathsf{cc}(G, T', 2d_0) \right) \\
&\leq c_{1,\delta} N k^{-b_1+3b_0} + c_{1,\delta} k^{b_1} \left(\max_{T' \subseteq V \setminus S_1} \mathsf{cc}(G, T', 2d_0) \right)
\end{aligned}$$

Where the final inequality holds by setting $c_{1,\delta} = \max\{12\alpha, 1\}$ the base case holds when $i = 1$.

Now, for the sake of induction, assume that $\mathsf{cc}(G, T, 2d_0) \leq c_{i,\delta} N k^{3-b_1} + c_i \sum_{j=1}^{i} N k^{b_1+\ldots+b_{i-2}+4b_{i-1}-b_i} + c_{i,\delta} k^{b_1+\ldots+b_i} \left(\max_{T' \subseteq V \setminus S_i} \mathsf{cc}(G, T', 2d_i)\right)$ for some constant c_i. Now we can invoke Lemma 6 to argue that for any $T' \subseteq V \setminus S_i$ we have $\mathsf{cc}(G - S_i, T', 2d_i) \leq 12 e_{i+1} d_i + \frac{N}{e_i} \left(\max_{T'' \subseteq V \setminus S_{i+1}} \mathsf{cc}(G - S_{i+1}, 2d_{i+1})\right)$.

Thus, for any $T'' \subseteq V \setminus S_{i+1}$ we have

$$\begin{aligned}
\mathtt{cc}(G,T,2d_0) &\leq c_{i,\delta} N k^{3-b_1} + c_{i,\delta} \sum_{j=1}^{i} N k^{b_1+\ldots+b_{i-2}+4b_{i-1}-b_i} \\
&\quad + c_{i,\delta} k^{b_1+\ldots+b_i} \left(12 e_{i+1} d_i + \frac{N}{e_i} \left(\max_{T' \subseteq V \setminus S_i} \mathtt{cc}(G - S_i, T', 2d_i) \right) \right) \\
&\leq c_{i,\delta} N k^{3-b_1} + c_{i,\delta} \sum_{j=1}^{i} N k^{b_1+\ldots+b_{i-2}+4b_{i-1}-b_i} \\
&\quad + c_{i,\delta} k^{b_1+\ldots+b_i} \left(12 e_{i+1} d_i + \frac{N}{e_{i+1}} \max_{T'' \subseteq V \setminus S_{i+1}} (\mathtt{cc}(G - S_{i+1}, T'', 2d_{i+1})) \right) \\
&\leq c_{i,\delta} N k^{3-b_1} + c_{i,\delta} \sum_{j=1}^{i} N k^{b_1+\ldots+b_{i-2}+4b_{i-1}-b_i} \\
&\quad + 12 c_{i,\delta} k^{b_1+\ldots+b_i} N k^{-b_{i+1}} k^{3b_i} + \\
&\quad c_{i,\delta} k^{b_1+\ldots+b_i+b_{i+1}} \left(\max_{T'' \subseteq V \setminus S_{i+1}} \mathtt{cc}(G - S_{i+1}, T'', 2d_{i+1}) \right) \\
&\leq c_{i+1,\delta} N k^{3-b_1} + c_{i,\delta} \sum_{j=1}^{i} N k^{b_1+\ldots+b_{i-2}+4b_{i-1}-b_i} \\
&\quad + c_{i+1,\delta} k^{b_1+\ldots+b_i} N k^{-b_{i+1}} k^{3b_i} + \\
&\quad c_{i+1,\delta} k^{b_1+\ldots+b_i+b_{i+1}} \left(\max_{T'' \subseteq V \setminus S_{i+1}} \mathtt{cc}(G - S_{i+1}, T'', 2d_{i+1}) \right)
\end{aligned}$$

where the final inequality holds by setting $c_{i+1,\delta} = 12 c_{i,\delta}$. Thus the inductive claim holds. □

Reminder of Theorem 9. *Let G be a directed acyclic graph with $N = |V|$ nodes and maximum indegree δ. Suppose that for some constant $\alpha > 0$ the graph G is (e,d)-reducible with $d = \alpha N^3/e^3$ for any $e \geq 1$. Then there is a constant $c > 0$ such that for every $k \geq 1$ there is a set $S_0 \subseteq V$ of size $|S_0| \leq N/k$ such that $\mathtt{cc}(G - S_0) \leq cNk^{12/5} = O\left(Nk^{12/5}\right)$ Proof of Theorem 9.* Because the graph is $(e, \alpha N^3/e^3)$-reducible we can find a set S_0 of size N/k such that $d_0 = \mathsf{depth}(G - S_0) \leq \alpha k^3$. We can apply Lemma 11 with $T = \mathtt{sinks}(G - S_0)$ — here $\mathtt{sinks}(G - S_0) = \{v \leq N \ : \ \forall u \neq v.(v,u) \notin E(G - S_0)\}$ denotes the set of sink nodes of $G - S_0$. We have $\mathtt{cc}(G - S_0) \leq \mathtt{cc}(G - S_0, T, 2d_0)$. We can now apply Lemma 11 to bound

$$\begin{aligned}
\mathtt{cc}(G - S_0, T, 2d_0) \leq c_{i,\delta} N k^{3-b_1} + c_{i,\delta} \sum_{j=1}^{i} N k^{b_1+\ldots+b_{i-2}+4b_{i-1}-b_i} + \\
c_{i,\delta} k^{b_1+\ldots+b_i} \left(\max_{T' \subseteq V \setminus S_i} \mathtt{cc}(G - S_i, T', 2d_i) \right)
\end{aligned}$$

where we will set $b_1 = \frac{3}{5}$ and $b_{j+1} = b_j \left(\frac{3}{4}\right)$ for all $1 < j \leq i$. Here, S_i is set of size $|S_i| \leq e_i = Nk^{-b_i}$ such that $\mathsf{depth}(G - S_i) \leq d_i = \alpha k^{3b_i}$ so we trivially have $\mathsf{cc}(G - S_i, T', 2d_i) \leq N\alpha k^{3b_i}$. Therefore, the last term is at most $c_{i,\delta} N\alpha k^{b_1+\ldots+b_i+3b_i}$.

To upper bound 6, we will upper bound each of the three terms.

The first term is trivially upper bounded as $c_{i,\delta} Nk^{3-b_1} \leq c_{i,\delta} Nk^{12/5}$

For the other two terms, it will be helpful to upper bound $B_i \doteq \sum_{j=1}^{i} b_j \leq \sum_{j=1}^{\infty} b_1 \left(\frac{3}{4}\right)^j \leq \frac{b_1}{1-\frac{3}{4}} = 12/5$.

Now, let $Z_i \doteq 4b_{i-1} - b_i + \sum_{j=1}^{i-2} b_i$. Consider $Z_{j+1} - Z_j$, for any $j > 1$. It follows from our definition of Z_i that for all j we have

$$\begin{aligned} Z_{j+1} - Z_j &= \sum_{i=0}^{j-1} b_i + 4b_j - b_{j+1} - \left(\sum_{i=0}^{j-2} b_i + 4b_{j-1} - b_j \right) \\ &= 5b_j - 3b_{j-1} - b_{j+1} \\ &= (5 - (9/4) - 9\,(9/16))\, b_j \\ &= (35/16)\, b_j > 0 \ . \end{aligned}$$

Where the last inequality follows since $\forall j, b_j > 0$. This means the difference is positive and thus that Z_j is increasing for increasing j.

Recall that it follows from our definitions of Z_i and B_i that $Z_i = B_{i-2} + 4b_{i-1} - b_i$. Since $b_i \to 0$ as $i \to \infty$ it follows that

$$\lim_{i\to\infty} Z_i = \lim_{i\to\infty} B_{i-2} = \frac{12}{5} \ .$$

Since Z_j is increasing it follows that $Z_j \leq \frac{12}{5}$ for all j. Thus, it follows that the middle term is upper bounded as

$$c_{i,\delta} \sum_{j=1}^{i} Nk^{b_1+\ldots+b_{i-2}+4b_{i-1}-b_i} \leq c_{i,\delta} ik^{\frac{12}{5}} \ .$$

Similarly, we can define $Y_i = B_i + 3b_i$. We have

$$Y_{i+1} - Y_i = 4b_{i+1} - 3b_i = 4\left(\frac{3}{4} b_i\right) - 3b_i = 0 \ .$$

Since $Y_2 = b_1 + 4b_2 = 4b_1 = \frac{12}{5}$ we have $Y_i = \frac{12}{5}$ for all $i > 1$. Thus, the final final term is upper bounded as $c_{i,\delta} N\alpha k^{\frac{12}{5}}$ □

B.2 Proof of Theorem 10

Reminder of Theorem 10. *Let G be any directed acyclic graph with $N = |V|$ nodes and maximum indegree δ. There is a constant $c_\delta > 0$ such that for every*

$k \geq 1$ *there is a set* $S_0 \subseteq V$ *of size* $|S_0| \leq N/k$ *such that:*

$$\begin{cases} \mathsf{cc}(G - S_0) \leq \frac{N^2}{(\log^8 N)} & \text{if } k < \frac{c_\delta \log N}{8 \log\log N} \\ \mathsf{cc}(G - S_0) \leq \frac{c_\delta N^2 \log\log N}{\left(2^{\frac{\log N}{c_\delta k}} \log N\right)} & \text{otherwise} \end{cases}$$

Proof of Theorem 10. For any DAG G with constant indegree δ there is a constant $c_\delta > 0$ such that for any parameter $1 \leq z \leq \log_2 N$ the DAG G is $(c_\delta N z / \log N, N/2^z)$-reducible [AB16].

If $k \leq \frac{c_\delta \log N}{8 \log\log N}$, then we set $z_0 \doteq (8 \log\log N)/c_\delta \leq (\log N)/k$ and observe there is a set S_0 of size $|S_0| \leq \frac{c_\delta N z_0}{\log N} = \frac{N}{k}$ such that $\mathsf{depth}(G - S_0) \leq \frac{N}{2^{z_0}} = \frac{N}{\log^8 N}$. Therefore, we can pebble $G - S_0$ in parallel in *at most* $\mathsf{depth}(G - S_0) \leq \frac{N}{\log^8 N}$ rounds at cumulative cost *at most* $\mathsf{cc}(G - S_0) \leq N \times \mathsf{depth}(G - S_0) \leq \frac{N^2}{\log^8 N}$.

Suppose, on the other hand, that $k > \frac{c_\delta \log N}{8 \log\log N}$. We define $z_0 = (\log N)/(c_\delta k)$ and $z_1 = z_0 + 8 \log\log N$. We will set $d_i \doteq N/2^{z_i}$ and $e_i \doteq N z_i / \log N$ for $i \in \{0, 1\}$ and let S_0 be a set of $|S_0| \leq c_\delta N z_0 / \log N$ nodes such that $\mathsf{depth}(G - S_0) \leq d_0$. Similarly, let S_1 be a set of $|S_1| \leq c_\delta N z_1 / \log N$ nodes such that $\mathsf{depth}(G - S_1) \leq d_0$. Observe that $|S_0| \leq c_\delta N \left((\log N)/(c_\delta k)\right) / \log N = N/k$.

We further observe that $d_1 \leq d_0$ and $2 d_1 N \leq e_1 d_0$ since $2 d_{i+1} N = \frac{2N^2}{2^{z_{i+1}}} = \frac{2N^2}{2^{z_i + 8 \log\log N}} = \frac{2N^2}{\log^8 N 2^{z_i}} = \frac{2 N d_i}{\log^8 N} \leq e_{i+1} d_i$. The last inequality follows because $e_{i+1} = \frac{z_{i+1} N}{\log N} \geq \frac{2N}{\log N} \geq \frac{2N}{\log^8 N}$. Therefore we can apply lemma 6 to the graph $G - S_0$ with target set $T = V - S_0$. Then the pebbling cost is bounded as follows:

$$\begin{aligned}
\mathsf{cc}\left(G - S_0, T, 2d_0\right) &\leq (4\delta + 4) e_1 d_0 + \frac{N}{e_1} \left(\mathsf{cc}(G - S_1, T, 2d_1)\right) \\
&\leq (4\delta + 4) \frac{N}{2^{z_0}} \left(\frac{N \cdot 8 \log\log N}{\log N}\right) + \\
&\quad \frac{\log N}{8 \log\log N} \left(cc(G - S_1, T, 2d_1)\right) \\
&\leq (4\delta + 4) \frac{N}{2^{z_0}} \left(\frac{N \cdot 8 \log\log N}{\log N}\right) + \frac{\log N}{8 \log\log N} \left(N 2 d_1\right) \\
&\leq (4\delta + 4) \frac{N}{2^{z_0}} \left(\frac{N \cdot 8 \log\log N}{\log N}\right) + 2 \frac{\log N}{8 \log\log N} \left(\frac{N^2}{2^{z_1}}\right) \\
&\leq (4\delta + 4) \frac{N}{2^{z_0}} \left(\frac{N \cdot 8 \log\log N}{\log N}\right) + 2 \frac{\log N}{8 \log\log N} \left(\frac{N^2}{2^{z_0} \log^8 N}\right) \\
&\leq (4\delta + 4) \frac{N}{2^{z_0}} \left(\frac{N \cdot 8 \log\log N}{\log N}\right) + 2 \frac{N^2}{8 \log\log N \log^7 N 2^{z_0}} \\
&\leq c_\delta \frac{N^2 \log\log N}{2^{z_0} \log N} \\
&= c_\delta \frac{N^2 \log\log N}{2^{\frac{\log N}{c_\delta k}} \log N}
\end{aligned}$$

□

C Merkle Tree Reveal

In this appendix, we formally define the MerkleTreeReveal procedure, which enables a prover to efficiently prove knowledge of a leaf in a Merkle tree rooted at commitment τ. This proof consists of the leaf value and an authentication path of internal hash values that allows the verifier to recompute the Merkle root and confirm consistency with τ.

We assume that the Merkle tree is a full binary tree of depth n with $N = 2^n$ leaves $\ell_1, \ldots, \ell_N$, and that each internal node is labeled by applying a random oracle H to its children and a salt. The tree is constructed recursively as follows.

C.1 Merkle Tree Construction

Given input $\chi \in \{0,1\}^\lambda$ and a sequence of $N = 2^n$ leaves $\ell_1, \ldots, \ell_N$, we label each internal node as follows:

- Let τ_v denote the label of node $v \in \{0,1\}^{\leq n}$, where v is the binary string denoting the position in the tree. That is, $v = v_1 v_2 \ldots v_k$ for some $k \leq n$.
- For a leaf node $v \in \{0,1\}^n$, let $i = \texttt{index}(v)$ be the integer corresponding to v. Define $\tau_v = \ell_{i+1}$.
- For an internal node $v \in \{0,1\}^{<n}$, define its left and right children as $v0$ and $v1$, respectively, and set

$$\tau_v = H(\chi, \tau_{v0}, \tau_{v1}).$$

- The Merkle root is $\tau = \tau_\varepsilon$, where ε is the empty string (the root of the tree).

Merkle Tree Reveal

The MerkleTreeReveal procedure provides a proof that a leaf value ℓ_i appears at a specific index $i \in [0, N-1]$ in the tree with root τ.

Let $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ denote the binary representation of index i, i.e., $x = \texttt{bin}(i)$. Then the authentication path for ℓ_i consists of the sibling hashes needed to recompute the Merkle root.

Definition 15 (Merkle Tree Reveal). *Given an index $i \in [0, N-1]$ and a Merkle tree constructed over $\ell_1, \ldots, \ell_N$ with root $\tau = \tau_\varepsilon$, define the Merkle Tree Reveal μ_i as the tuple*

$$\mu_i = \text{MerkleTreeReveal}(x) = (\ell_i, \{\tau_{x^{(j)}}\}_{j=1}^n),$$

where:

- *$x = (x_1, \ldots, x_n) = \texttt{bin}(i)$ is the binary representation of index i,*

– $x^{(j)} = x_1 x_2 \dots x_{j-1} \bar{x}_j$ *denotes the sibling node at depth* j*, where* $\bar{x}_j$ *is the complement of bit* x_j*.*

That is, for each level j from the leaf to the root, the prover includes the label of the sibling node needed to recompute the parent label. Given ℓ_i and the authentication path $\{\tau_{x^{(j)}}\}_{j=1}^{n}$, the verifier can recursively compute the label $\tau_{x_1 \dots x_{n-j}}$ at each level, ending at τ_ε. If the computed root equals the claimed root τ, the verifier accepts the reveal as valid.

C.2 Merkle Tree Check

The MerkleTreeCheck procedure verifies that the output of a given MerkleTreeReveal is valid. Given input string χ, leaf index i, value ℓ_i, authentication path $\mu_i = \{\tau_{x^{(j)}}\}_{j=1}^{n}$, and the claimed Merkle root τ, MerkleTreeReveal$(\chi, i, \ell_i, \mu_i, \tau)$ proceeds as follows:

1. Initialize $\sigma_0 = \ell_i$.
2. For $j = 1$ to n, compute

$$\sigma_j = \begin{cases} H(\chi, \sigma_{j-1}, \tau_{x^{(j)}}), & \text{if } x_j = 0, \\ H(\chi, \tau_{x^{(j)}}, \sigma_{j-1}), & \text{if } x_j = 1. \end{cases}$$

3. If $\sigma_n = \tau$, then accept; otherwise, reject.