`Scrypt` is Maximally Memory-Hard

Joël Alwen[1], Binyi Chen[2], Krzysztof Pietrzak[1], Leonid Reyzin[3], and Stefano Tessaro[2]

[1] IST Austria
{jalwen,pietrzak}@ist.ac.at
[2] UC Santa Barbara
{binyichen,tessaro}@cs.ucsb.edu
[3] Boston University
reyzin@cs.bu.edu

Abstract. Memory-hard functions (MHFs) are hash algorithms whose evaluation cost is dominated by memory cost. As memory, unlike computation, costs about the same across different platforms, MHFs cannot be evaluated at significantly lower cost on dedicated hardware like ASICs. MHFs have found widespread applications including password hashing, key derivation, and proofs-of-work.

This paper focuses on `scrypt`, a simple candidate MHF designed by Percival, and described in RFC 7914. It has been used within a number of cryptocurrencies (e.g., Litecoin and Dogecoin) and has been an inspiration for Argon2d, one of the winners of the recent password-hashing competition. Despite its popularity, no rigorous lower bounds on its memory complexity are known.

We prove that `scrypt` is *optimally memory hard*, i.e., its cumulative memory complexity (cmc) in the parallel random oracle model is $\Omega(n^2 w)$, where w and n are the output length and number of invocations of the underlying hash function, respectively. High cmc is a strong security target for MHFs introduced by Alwen and Serbinenko (STOC '15) which implies high memory cost even for adversaries who can amortise the cost over many evaluations and evaluate the underlying hash functions many times in parallel. Our proof is the first showing optimal memory hardness for any MHF.

Our result improves both quantitatively and qualitatively upon the recent work by Alwen *et al.* (EUROCRYPT '16) who proved a *weaker* lower bound of $\Omega(n^2 w/\log^2 n)$ for a *restricted* class of adversaries.

Keywords: Scrypt, memory-hard functions, password hashing.

1 Introduction

Several applications rely on so-called "moderately-hard tasks" that are not infeasible to solve, but whose cost is non-trivial. The cost can for example be the hardware or electricity cost of computation as in proofs of work [DN93,JJ99,DNW05] or time-lock puzzles [RSW96], the cost for disk storage space as in proofs of space [DFKP15], the cost of "human attention" in captchas [vABHL03] or the cost of memory in memory-bound [DGN03,ABW03] or memory-hard functions [Per09,AS15], the latter being the topic of this work. Applications of such tasks include the prevention of spam [DN93], protection against denial-of-service attacks [JB99], metering client access to web sites [FM97], consensus protocols underlying decentralized cryptocurrencies [Cha11] or password hashing [PHC], which we'll discuss in more detail next.

In the setting of *password hashing*, a user's password (plus a salt and perhaps other system-dependent parameters) is the input to a moderately-hard function f, and the resulting output is the password hash to be kept in a password file. The hope is that even if the password file is compromised, a brute-force dictionary attack remains costly as it would require an attacker to evaluate f on every password guess. Traditional approaches for password hashing have focused on iterating a hash function a certain number (typically a few thousands) of times, as for instance in PBKDF2. An advantage for the honest user results from the fact that he or she needs to compute f only once on the known password, while an attacker is forced to compute f on a large number of passwords. However, this advantage can be eroded, because in constrast to honest users, who typically use general-purpose hardware, attackers may invest into special-purpose hardware like ASICs (Application Specific Integrated Circuits) and recoup the investment over multiple evaluations. Moreover, such special-purpose hardware may exploit parallelism, pipelining, and amortization in ways that the honest user's single evaluation of f cannot. Consequently, the adversary's cost per evaluation can be several orders of magnitude lower than that for the honest user.

Memory-Hard Functions. To reduce the disparity between per-evaluation costs of the honest user and a potential attacker, Percival [Per09] suggested measuring cost by the amount of space used by the algorithm multiplied by the amount of time. A *memory-hard function* (MHF), in Percival's definition, is one where this evaluation cost measure is high not only for the honest user's sequential algorithm, but also no parallel algorithm can do much better. In particular, if a parallel algorithm can cut the time to evaluate f by some factor, it must come at the cost of increase in space by roughly the same factor. Since memory is inherently general-purpose, measuring cost in terms of space provides a reasonably accurate comparison of resources used by different implementations. We stress that memory hardness is a very different notion than that of memory-*bound* functions proposed by Abadi, Burrows, Manasse, and Wobber [ABW03,ABMW05], which maximize the number of memory accesses at unpredictable locations so that the inherent memory-access latency (resulting from frequent cache misses) imposes a lower bound on the *time* needed to evaluate the function which is independent from the actual CPU power. Memory hardness also does not guarantee that a lot of memory will be required, because it allows trading memory for time.

Alwen and Serbinenko [AS15] observed that Percival's notion of cost is not robust to amortization: it may be that an algorithm uses a large amount of memory at its peak, but a much smaller amount on average; pipelining multiple evaluations in such a way that peaks occur at different times can thus reduce the per-evaluation cost of f. They propose the notion of *cumulative memory complexity* (abbreviated $\mathsf{cc}_{\mathsf{mem}}$), which is robust to amortization. It is defined as the sum of memory actually used at each point in time (rather than the product of peak memory and time). We will use this notion in our work. The $\mathsf{cc}_{\mathsf{mem}}$ of a function is defined as the lowest $\mathsf{cc}_{\mathsf{mem}}$ of all algorithms that evaluate the function.

Best-possible hardness. Given the state of computational complexity theory, where we cannot even prove superlinear lower bounds for problems in NP, all $\mathsf{cc}_{\mathsf{mem}}$ lower-bound results so far necessarily make use of idealized models of computation, like the random oracle model.

Many candidate memory-hard functions (including `scrypt`) can be viewed as a mode of operation for an underlying building block like a cryptographic hash-function h. Such MHFs come with an evaluation algorithm — which we'll call "the naïve algorithm" — which makes only sequential access to h. Note that after t steps (each involving at most one query to h), even a naïve algorithm which stores all t outputs it received from $\mathsf{h} : \{0,1\}^* \to \{0,1\}^w$ will not use more that $O(t \cdot w)$ memory; therefore, if the naïve algorithm for an MHF f^{h} runs for n steps total, the $\mathsf{cc}_{\mathsf{mem}}$ of f^{h} will be in $O(n^2 \cdot w)$. Thus a lower bound on the $\mathsf{cc}_{\mathsf{mem}}$ of $\Omega(n^2 \cdot w)$ is the best we can hope for in any model which captures at least the naïve algorithm.

If the naïve algorithm has the feature that its memory access pattern (addresses read and written) is independent of the input, the MHF is called *data-independent*. A data-independent f^{h} can be represented as a directed graph, with a unique source corresponding to the input, a unique sink corresponding to the final output, and the other nodes indicating intermediary values where the value of a node is computed as a function of the nodes of its parents (using one invocation of h). To derive meaningful bounds, we require that this graph has constant in-degree, so computing an intermediate value takes constant time. The evaluation of f^{h} can now be cast as a graph pebbling problem [AS15]. Any constant in-degree graph can be pebbled (in the so called parallel black pebbling game) using "cumulative pebbling complexity" $\mathsf{cc}_{\mathsf{peb}} = O(n^2/\log n)$.[4] As any such pebbling implies an evaluation algorithm with $\mathsf{cc}_{\mathsf{mem}} \approx \mathsf{cc}_{\mathsf{peb}} \cdot w$, we get an $O(w \cdot n^2/\log n)$ upper bound on $\mathsf{cc}_{\mathsf{mem}}$ for any data-indepedent MHF [AB16]. On the positive side, [AS15] design a data-independent function with $\mathsf{cc}_{\mathsf{mem}} = \Omega(w \cdot n^2/\log^{10}(n))$ in the parallel random oracle model. This calls for the question of whether the lower bound of $\Omega(w \cdot n^2)$ is achievable at all; the above discussion shows that to achieve this lower bound, it will not be sufficient to only consider data-independent MHFs.

The `Scrypt` MHF. Percival [Per09] proposed a candidate (data-dependent) memory-hard function called `scrypt`.[5] On input X, the $\mathtt{scrypt}^{\mathsf{h}}$ function– where h is a cryptographic hash function modeled as a random oracle for the lower

[4] Technically, the bound is marginally worse, $O(n^2/\log^{1-\epsilon}(n))$ for any $\epsilon > 0$

[5] In fact, what we discuss in the following is Percival's ROMix construction, which constitutes the core of the actual `scrypt` function. We use the two names interchangeably.

bound proof – computes values $X_0, X_1, \ldots, X_{n-1}, S_0, \ldots, S_n$ as defined below, and finally outputs S_n

- $X_0 = X$ and for $i = 1, \ldots, n-1 \ : \ X_i = \mathsf{h}(X_{i-1})$
- $S_0 = h(X_{n-1})$ and for $i = 1, \ldots, n \ : \ S_i = \mathsf{h}(S_{i-1} \oplus X_{S_{i-1} \bmod n})$

`Scrypt` has found widespread popularity: it is used the proofs-of-work schemes for cryptocurrencies (most notably Litecoin [Cha11], but also Tenebrix or Dogecoin), is described by an RFC [PJ16], and has inspired the design of one of the Password-hashing Competition's [PHC] winners, Argon2d [BDK16].

An intuitive explanation for why `scrypt` was conjectured to be memory-hard is as follows. View the first portion of `scrypt` as an n-node line graph, with nodes labeled by $X_0, \ldots, X_{n-1}$. To compute S_{i+1}, an algorithm needs $X_{S_i \bmod n}$, whose index ($S_i \bmod n$) is random and unknown until S_i is computed. If the algorithm stores a subset of the X values of size p before S_i is known, then the label of a random node in the line graph will be on average $n/(2p)$ steps from a stored label, and will therefore take $n/(2p)$ sequential evaluations of h compute, for a total memory·time cost of $p \cdot n/(2p) = n/2$. Since there are n S_i values to compute, this strategy has $\mathsf{cc}_{\mathsf{mem}}$ of $w \cdot n \cdot n/2 = \frac{1}{2}n^2$.

This simple argument, however, does not translate easily into a proof. The two main challenges are as follows. First, in general an algorithm computing `scrypt` is not restricted to just store labels of nodes, but can compute and store arbitrary information. Surprisingly, f for which storing information other than just labels provably decreases $\mathsf{cc}_{\mathsf{mem}}$ have been constructed in [ACK$^+$16a, Appendix A]. Second, an algorithm is not compelled to keep all p labels in memory after the index $S_i \bmod n$ is known. In fact, [AS15] show that if an algorithm is given the indices $S_i \bmod n$ in advance, it's possible to evaluate $\mathtt{scrypt}^{\mathsf{h}}$ with $\mathsf{cc}_{\mathsf{mem}}$ only $O(w \cdot n^{1.5})$.

Previous work on `scrypt`. Percival's original paper [Per09] proposed an analysis of `scrypt`, but his analysis is incorrect, as we point out in Appendix A, in addition to not targeting $\mathsf{cc}_{\mathsf{mem}}$. Recent progress toward proving that `scrypt` is memory-hard was made by Alwen et al. [ACK$^+$16b]. They lower bound the $\mathsf{cc}_{\mathsf{mem}}$ of `scrypt` by $\Omega(w \cdot n^2/\log^2 n)$, albeit only for a somewhat restricted class of adversaries (informally, adversaries who can store secret shares of labels, but not more general functions). We'll compare their work with ours in more detail below.

Our Results. We give the first non-trivial unconditional lower bound on $\mathsf{cc}_{\mathsf{mem}}$ for $\mathtt{scrypt}^{\mathsf{h}}$ in the parallel random oracle model, and our bound already achieves optimal $\mathsf{cc}_{\mathsf{mem}}$ of $\Omega(w \cdot n^2)$.

We'll give the exact theorem statement and an overview of the proof in Section 3. However, to appreciate the novelty of our results, we note that the only existing proofs to lower bound $\mathsf{cc}_{\mathsf{mem}}$ of MHFs go through some kind of lower bounds for pebbling.

For *data independent* MHFs [AS15] there is an elegant argument (known as "ex post facto") stating that a lower bound on the cumulative complexity for the parallel black pebbling game translates directly into a lower bound for $\mathsf{cc}_{\mathsf{mem}}$.

Thus, the problem is reduced to a purely combinatorial problem of proving a pebbling lower bound on the graph underlying the MHF.

For *data dependent* MHFs no such result, showing that pebbling lower bounds imply $\mathsf{cc}_{\mathsf{mem}}$ lower bounds for general adversaries, is known.[6] The lower bound on $\mathsf{cc}_{\mathsf{mem}}$ for `scrypt` from [ACK+16b] was also derived by first proving a lower bound on the pebbling complexity, but for a more powerful pebbling adversary that can use "entangled" pebbles. This lower bound then translated into a lower bound for $\mathsf{cc}_{\mathsf{mem}}$ for a limited class of adversaries who, apart from labels, can also store "secret shares" of labels. It was conjectured [ACK+16b] that lower bounds for this entangled pebbling game already imply lower bounds on $\mathsf{cc}_{\mathsf{mem}}$ for *arbitrary* adversaries, and a combinatorial conjecture was stated which, if true, would imply this. Unfortunately the strongest (and simplest) version of this conjecture has already been refuted. A weaker version of the conjecture has been "weakly" refuted, in the sense that, even if it was true, one would lose a factor of at least $\log(n)$ by going from pebbling to memory lower bounds. (The current state of the conjecture is available on the eprint version of the paper [ACK+16a].)

In this work, in Section 5, we also prove an optimal $\Omega(n^2)$ lower bound on the parallel cumulative pebbling complexity for a game which abstracts the evaluation of `scrypt`: we consider a path of length n, and an adversary must pebble n randomly chosen nodes on this graph, where the ith challenge node is only revealed once the node of challenge $i-1$ is pebbled. This already gives an optimal $\Omega(n^2 \cdot w)$ lower bound on $\mathsf{cc}_{\mathsf{mem}}$ for `scrypt` for adversaries who are only allowed to store entire labels, but not any functions thereof. This improves on the $\Omega(n^2/\log^2(n))$ lower bound from [ACK+16b], who use a rather coarse potential argument which roughly states that, for any challenge, either we pay a lot for pebbling the next challenge node, or the "quality" of our pebbling configuration decreases. As this quality cannot decrease too many times, at least every $\log(n)$'th challenge will cost $n/\log(n)$ in cumulative complexity, giving the overall $\Omega(n^2/\log^2(n))$ lower bound after n challenges. In this work we introduce a new technique for analyzing the cumulative pebbling cost where — for every challenge — we take into account the cumulative cost of the pebbling configurations *before* this challenge is revealed. Both the potential argument from [ACK+16b], as well as our new proof, rely on the generalization of the fact that given a configuration with p pebbles, and a random challenge, with good probability (say at least $\frac{1}{2}$), an adversary also needs also at least (roughly) n/p steps to pebble the challenge.

As discussed above, pebbling lower bounds are not known to directly imply $\mathsf{cc}_{\mathsf{mem}}$ lower bounds for data dependent MHFs, so to prove our main result in Section 6, we in some sense emulate our proof for the pebbling game directly in the parallel random oracle model. However, there are two challenges we will need to overcome. The first is that the adversary's state is not made of labels

[6] A lower bound on the parallel cumulative pebbling complexity is only known to imply a lower bound on $\mathsf{cc}_{\mathsf{mem}}$ for a very restricted class of adversaries who are allowed to store only labels, but not any function thereof.

(corresponding to pebbles), but could be any function thereof. Still, we will want to show that in order to compute a challenge, an adversary storing any $p \cdot w$ bits of information about the random oracle, will need to take with good probability (say at least $\frac{1}{2}$) at least (roughly) n/p steps. We will show this using a careful compression argument in Section 4. The second is the fact that in `scrypt` the challenges are not randomly and externally generated, but come from the random oracle.

2 Preliminaries

We review basic notation and concepts from the literature on memory-hard functions. We will also define the `scrypt` function as needed further below.

The parallel-random oracle model. We first define the parallel random-oracle model (pROM), essentially following the treatment from [AS15], with some highlighted differences.

Concretely, we consider an oracle-aided deterministic[7] algorithm A which runs in rounds, starting with round 1. Let h denote an oracle with w-bit outputs. It does not matter for our model whether oracle inputs are restricted in length, but it will be simpler to assume a general upper bound (even very large) on the length of its inputs to make the set of oracles finite.

In general, a *state* is a pair $(\tau, \mathbf{s})$ where *data* τ is a string and $\mathbf{s}$ is a tuple of strings. In an execution, at the end of round i, algorithm A produces as output an *output state* $\bar{\sigma}_i = (\tau_i, \mathbf{q}_i)$ where $\mathbf{q}_i = [q_i^1, \ldots, q_i^{z_i}]$ is a tuple of *queries* to h. At the begining of next round $i+1$, algorithm A gets as input the corresponding *input state* $\sigma_i = (\tau_i, \mathsf{h}(\mathbf{q}_i))$ where $\mathsf{h}(\mathbf{q}_i) = [\mathsf{h}(q_i^1), \ldots, \mathsf{h}(q_i^{z_i})]$ is the tuple of *responses* from h to the queries $\mathbf{q}_i$. In particular, since A is deterministic, for a given h the input state σ_{i+1} is a function of the input state σ_i.

The initial input state σ_0 is normally empty with length 0 (though in the proof we will also need to consider a non-empty initial input state); an input X is given together with σ_0 in the first round. We require that A eventually terminates and denote its output by $A^{\mathsf{h}}(X)$.

Complexity measure. For a given execution the complexity measure we are going to be concerned with is the sum of the bit-lengths of the input states. To that make this precise we introduce the following notation. For a string x we denote its bit-length by $|x|$. For state $\sigma = (\tau, \mathbf{s})$ where $\mathbf{s} = [s_1, \ldots, s_z]$ we denote the bit-length (or size) of σ by $|\sigma| = |\tau| + \sum_{j=1}^{z} |s_j|$. We can now define the *cumulative (memory) complexity* of an execution of algorithm A on input X using oracle h resulting in *input* states $\sigma_0, \sigma_1, \ldots$ as

$$\mathsf{cc}_{\mathsf{mem}}(A^{\mathsf{h}}(X)) = \sum_{i \geq 0} |\sigma_i| \ .$$

[7] Considering deterministic algorithms is without loss of generality as we can always fix the randomness of A to some optimal value.

We will assume without loss of generality that at each round, the query tuple $\mathbf{q}$ contains at least on query, for otherwise A can proceed directly to the next round where it issues a query, without increasing its cumulative complexity. In particular, this implies $|\sigma_i| \geq w$ for $i > 0$.

Note that $\mathsf{cc}_{\mathsf{mem}}$ does not charge anything for computation or memory used within each round itself. We are also allowing inputs to h to be arbitrary long without extra memory cost — only the output length w is charger to the cumulative complexity. This only makes our *lower bound* stronger. Note however that $\mathsf{cc}_{\mathsf{mem}}$ is good for upper bounds only when computation is dominated by the memory cost (as is the case for the naïve evaluation algorithm of $\mathtt{scrypt}^{\mathsf{h}}$, which, aside from querying h sequentially, performs only trivial computations, such as exlusive-ors and modular reductions).

The $\mathtt{scrypt}$ MHF. We will consider the $\mathtt{scrypt}^{\mathsf{h}}$ function throughout this paper (more specifically, we study its core, ROMix, as defined in [Per09]). Recall that for a hash function $\mathsf{h} : \{0,1\}^* \to \{0,1\}^w$, $\mathtt{scrypt}^{\mathsf{h}}$ on input $X \in \{0,1\}^w$ and parameter $n \in \mathbb{N}$ computes values $X_0, X_1, \dots, X_{n-1}, S_0, \dots, S_n$ and outputs S_n, where

- $X_0 = X$ and for $i = 1, \dots, n-1 \ : \ X_i = \mathsf{h}(X_{i-1})$
- $S_0 = h(X_{n-1})$ and for $i = 1, \dots, n \ : \ S_i = \mathsf{h}(S_{i-1} \oplus X_{S_{i-1} \bmod n})$

We will also define intermediate variables $T_0, \dots, T_n$ with $T_0 = X_{n-1}$ and $T_i = S_{i-1} \oplus X_{S_{i-1} \bmod n}$ for $1 \leq i \leq n$, so that $S_i = \mathsf{h}(T_i)$.

Note that one may not want to restrict X to w bits. In this case, one can replace X with $\mathsf{h}(X)$ in the above construction. For notational simplicity, we will only analyze the w-bit input case in this paper, but the general analysis is very similar.

Graph and pebbling preliminaries. For some of our partial results below, we will adopt the graph-pebbling view on computing candidate MHFs, following [AS15]. A (parallel black) pebbling considers an acyclic graph $G = (V, E)$. At each time step t starting with $t = 0$, the adversary maintains a subset P_t of nodes ("pebbles"). A node v is allowed (but not required) to get a pebble at time t if there is a pebble on all of its predecessors (i.e., all v' such that $(v', v) \in E$), or if there was a pebble on v itself at time $t-1$. Formally, define $\mathsf{pre}(v)$ to be the set of all predecessors of v, and for $U \subseteq V$, define $U^+ = \{v \in V \ : \ \mathsf{pre}(v) \subseteq U\}$. Then, at time $t > 0$, the set P_t must be a subset of $P_{t-1} \cup P_{t-1}^+$.

We will use the notation $\mathsf{p}_i = |P_i| \geq 1$. The (parallel) *cumulative pebbling complexity* of a sequence of pebbling configuration $P_0, P_1, \dots, P_t$ is $\sum_{i=0}^t \mathsf{p}_i$. We remark that we modify the pebbling rules slightly from [AS15] by not permitting the adversary to put a pebble on the source for free: v_0 is contained in P_0 and cannot be added to P_t if it is absent in P_{t-1} (this change will simplify calculations, and only increase the size of each set by 1).

Pebbling with challenges. Normally, the goal of pebbling games is to place a pebble on the sink of the graph. Here, we are going to consider pebbling games with Q *challenges* on a graph $G = (V, E)$, where the adversary proceeds in rounds, and in each round i, it receives a random challenge $c_i \in V$ (usualy uniform from a subset $V' \subseteq V$), and the goal it to place a pebble on c_i, which enables the adversary to move to the next round (unless this was the last challenge c_Q, in which case the game terminates.) For instance, the core of the evaluation of `scrypt` is captured by the *line graph* with vertices $v_0, \dots, v_{n-1}$ and edges (v_i, v_{i+1}) for $i = 0, \dots, n-2$, and we will study this pebbling game in detail below.

3 Main Result and Overview

In this section, we state our main result, and give a brief high-level overview of the next sections.

Theorem 1 (Memory-hardness of `Scrypt`, main theorem). *For any $X \in \{0,1\}^w$ and $n \geq 2$, if $A^{\mathsf{h}}(X, n)$ outputs $S_n = \mathtt{scrypt}^{\mathsf{h}}(X, n)$ with probability χ, where the probability is taken over the choice of the random oracle* h*, then with probability (over the choice of* h*) at least $\chi - .08n^6 \cdot 2^{-w} - 2^{-n/20}$,*

$$\mathsf{cc}_{\mathsf{mem}}(A^{\mathsf{h}}(X)) > \frac{1}{25} \cdot n^2 \cdot (w - 4\log n) \ .$$

We note that if w is large enough in terms of n (say, $4 \log n \leq w/2$, which clearly holds for typical values $w = 256, n = 2^{20}$), then $\mathsf{cc}_{\mathsf{mem}}(A^{\mathsf{h}}(X))$ is in $\Omega(n^2 w)$. As discussed in the introduction, this is best possible up to constant factors, as already the (sequential) naïve algorithm for evaluating $\mathtt{scrypt}^{\mathsf{h}}$ has $\mathsf{cc}_{\mathsf{mem}} \in O(w \cdot n^2)$.

PROOF OUTLINE. The proof consists of three parts outlined below. The first two part, in fact, will give rise to statements of independent interest, which will then be combined into the proof of our main theorem.

- *Section 4: Single-shot time complexity.* To start with, we consider a pROM game where the adversary $A^{\mathsf{h}}(X)$ starts its execution with input X and an M-bit state σ_0 that can depend *arbitrarily* on h and X. Then, $A^{\mathsf{h}}(X)$ is given a random challenge $j \in \{0, \dots, n-1\}$ and must return $X_j = \mathsf{h}^j(X)$.
  Clearly, σ_0 may contain X_j, and thus in the best case, A may answer very quickly, but this should not be true for all challenges. We will prove a lower bound on the *expected* time complexity (in the pROM) of answering such a challenge. We will show that with good probability (e.g., $\frac{1}{2}$) over the choice of j, $A^{\mathsf{h}}(X)$ needs at least (roughly) nw/M steps.
  This validates in particular the intuition that the adversary in this game cannot do much better than an adversary in the corresponding pebbling game on the line graph with vertices $v_0, v_1, \dots, v_{n-1}$, where the adversary gets to choose an initial configuration with $p = M/w$ pebbles, and is then asked to put a pebble on v_j for a random $j \in \{0, 1, \dots, n-1\}$. Here, one can show that at least n/p steps are needed with good probability. In fact, this pebbling game is equivalent to a variant of the above pROM game where the adversary only stores random-oracle output labels, and thus our result shows that an adversary cannot do much better than storing whole labels.

- *Section 5: Multi-challenge cumulative pebbling complexity.* In the above scenario, we have only considered the *time* needed to answer a challenge. There is no guarantee, a priori, that the cumulative complexity is also high: An optimal adversary, for instance, stores p labels corresponding to equidistant pebbles, and then computes the challenge from the closest label, dropping the remainder of the memory contents.

  Here, for the randomized pebbling game with Q challenges on the line graph, we will show a lower bound of $\Omega(nQ)$ on the cumulative pebbling complexity. Our argument will use in particular a (generalization) of the above single-shot trade-off theorem, i.e., the fact that whenever p pebbles are placed on the line, at least n/p steps are needed with good probability to pebble a randomly chosen node. We will use this to lower bound the cumulative complexity *before* each particular challenge is answered. Our proof gives a substantial quantitative improvement over the looser lower bound of [ACK+16b].
- *Section 6:* $\mathsf{cc}_{\mathsf{mem}}$ *of* `scrypt`. Finally, we lower bound the cumulative memory complexity of $\texttt{scrypt}^{\mathsf{h}}$ as stated in Theorem 1. Unfortunately, this does not follow by a reduction from the pebbling lower bound directly. Indeed, as discussed in the introduction (and as explained in [ACK+16b]), unlike for data-independent MHFs, for data-dependent MHFs like `scrypt` it is an open problem whether one can translate lower bounds on the cumulative *pebbling* complexity to lower bounds for cumulative *memory* complexity. Fortunately, however, through a careful analysis, we will be able to employ the same arguments as in the proof of Section 5 in the pROM directly.

  In particular, we will use our result from Section 4 within an argument following the lines to that of Section 5 in the pROM. One particularly delicate technical issue we have address is the fact that in $\texttt{scrypt}^{\mathsf{h}}$ the challenges are not sampled randomly, but will depend on the random oracle h, which the adversary can query. We will provide more intuition below in Section 6.

4 Time Complexity of Answering a Single Challenge in the Parallel Random Oracle Model

We prove the following theorem, and below discuss briefly how this result can be extended beyond the setting of `scrypt`.

Fix positive integers n, u and w, a string $X \in \{0,1\}^u$, a finite domain $\mathcal{D}$ that contains at least $\{X\} \cup \{0,1\}^w$, and let $\mathcal{R} = \{0,1\}^w$. Given a function $\mathsf{h} : \mathcal{D} \to \mathcal{R}$, define $X_i = \mathsf{h}^i(X)$. Let A be any oracle machine (in the parallel random oracle model as defined in Section 2) that on any input and oracle makes at most $q-1$ total queries to its oracle. Suppose $A^{\mathsf{h}}(X, j)$ starts on input state σ_0 with the goal of eventually querying X_j to h. Let t_j be the number of the earliest round in which $A^{\mathsf{h}}(X, j)$ queries X_j to h (with $t_j = \infty$ if never). We show that A cannot do much better than if it were doing the following in the corresponding random challenge pebbling game on the line graph: initially placing $p \approx M/w$ equidistant pebbles, and then pebbling the challenge from the closest pebble preceding it.

Theorem 2 (Single-Challenge Time Lower Bound). *There exists a set of random oracles* $\mathsf{good_h}$ *such that* $\Pr_{\mathsf{h}\in\mathcal{R}^{\mathcal{D}}}[\mathsf{h} \notin \mathsf{good_h}] \le qn^3 2^{-w}$*, and for every* $\mathsf{h} \in \mathsf{good_h}$*, the following holds: for every memory size* M*, and every input state* σ_0 *of length at most* M *bits,*

$$\Pr_{j \leftarrow \{0,\ldots,n-1\}}\left[t_j > \frac{n}{2p}\right] \ge \frac{1}{2},$$

where the probability is taken over only the challenge j *and* $p = \lceil (M+1)/(w - 2\log n - \log q) + 1 \rceil$*.*

We will actually prove a slightly more general result: for any $0 \le \mathrm{pr_{hard}} \le 1$,

$$\Pr_{j \leftarrow \{0,\ldots,n-1\}}\left[t_j > \frac{n(1-\mathrm{pr_{hard}})}{p}\right] \ge \mathrm{pr_{hard}}\,.$$

Proof. Recall that for each j, A performs t_j rounds of the following process. At round k read an input state containing oracle responses $\mathsf{h}(\mathbf{q}_{k-1})$ (except for $k = 1$, when A reads σ_0). Then (after arbitrary computation) produce an output state containing oracle queries $\mathbf{q}_k$. We count rounds starting from 1. Consider the sequence of such tuples of queries and responses to and from h. If the first appearance of X_i in this sequence is as a query to h in round k (that is $k > 0$ is minimal such that $X_i \in \mathbf{q}_k$), then we assign X_i position $\pi_{ij} = k$. If instead the first appearance of X_i is as a response from h to query X_{i-1} made at round k (that is $k > 0$ is minimal such that $X_{i-1} \in \mathbf{q}_k$), then we assign X_i position $\pi_{ij} = k + 1/2$. In all other cases (i.e., if X_i does not appear in this sequence, or appears only because of a hash collision in response to some query that is not equal to X_{i-1}), let $\pi_{ij} = \infty$.

Let "best position" correspond to the earliest time, over all j, that X_i appears during the computation of X_j: $\beta_i := \min_j \pi_{ij}$; let "best challenge" $\mathsf{bestchal}_i$ be $\mathrm{argmin}_j\, \pi_{ij}$ (if argmin returns a set, pick one element arbitrarily). Let i be "blue" if β_i is an integer (that is, it was produced "out of the blue" by A as a query to h).

Let $B = \{i \text{ s.t. } i > 0 \text{ and } i \text{ is blue}\}$ (that is, all the blue indices except X_0). In the rest of the proof, we will show that the size of B cannot exceed $p - 1$ (for most h), where p, as defined in the theorem statement, is proportional to the memory size M; and that the amount of time to answer the challenge is at least its distance from the preceding blue index. Thus, blue indices effectively act like pebbles, and the bounds on the time to reach a random node in the line graph by moving pebbles apply.

Claim 1 *Given adversary* A *and input* X*, there exists a predictor algorithm* $\mathcal{P}$ *(independent of* h*, but with oracle access to it) with the following property: for every* h*, every* M*, and every length* M *input state* σ_0 *of* A*, there exists a hint of length* $|B|(2\log n + \log q)$ *such that given* σ_0 *and the hint,* $\mathcal{P}$ *outputs every* X_i *for* $i \in B$ *without querying* X_{i-1} *to* h*.*

Moreover, if we want fewer elements, we can simply give a shorter hint: there exists a predictor algorithm that similarly outputs p *elements of* B *whenever* $p \le |B|$*, given* σ_0 *and an additional* $p(2\log n + \log q)$*-bit hint.*

Note that the inputs to $\mathcal{P}$ can vary in size; we assume that the encoding of inputs is such that the size is unambiguous.

Proof. We will focus on the first sentence of the claim and address the second sentence at the end.

$\mathcal{P}$ depends on input label $X = X_0$ and algorithm A (which are independent of h). $\mathcal{P}$ will get the state σ_0 of A (which may depend on h) as input, and, for every $i \in B$, a hint containing the challenge $\mathsf{bestchal}_i$ for which X_i appears earliest, and the sequential order (among all the $q-1$ queries A makes in answering $\mathsf{bestchal}_i$) of the first query to X_i (using the value q to indicate that this query never occurs). This hint (which depends on h) will thus consist of a list of $|B|$ entries, each containing $i \in B$, $\mathsf{bestchal}_i$, and $\log q$ bits identifying the query number, for a total of $|B|(2\log n + \log q)$ bits.

$\mathcal{P}$ will build a table containing X_i for $i \geq 0$ (initializing $X_0 = X$). To do so, $\mathcal{P}$ will run A on every challenge in parallel, one round at a time. After each round k, $\mathcal{P}$ will obtain, from the output states of A, all the queries A makes for all the challenges in round k. Then $\mathcal{P}$ will fill in some spots in its table and provide answers to these queries as input states for round $k+1$ by performing the following three steps:

Step k. put any blue queries into its table (blue queries and their positions in the table can easily be recognized from the hint);

Step k+1/4. answer any query that can be answered using the table (i.e., any query that matches X_{i-1} in the table for some filled-in positions $i-1$ and i);

Step k+1/2. send remaining queries to h, return the answers to A, and fill in any new spots in the table that can be filled in (i.e., for every query that matches X_{i-1} in the table for some filled-in position $i-1$, fill in position i with the answer to that query).

Once every X_i for $i \in B$ is in the table, $\mathcal{P}$ queries h to fill in the missing positions in the table, if any, and outputs the prediction that $\mathsf{h}(X_{i-1}) = X_i$ for $i \in B$.

To prove that $\mathcal{P}$ simulates h correctly to A, it suffices to show that the table contains correct labels. This can be easily argued by induction on i. Assume all the labels in the table are correct up now. A new label X_i enters the table either because it is marked as blue (and thus correct by the hint) or is obtained as an answer from h to the query that $\mathcal{P}$ identified as X_{i-1} using the table (which is correct by inductive hypothesis).

The above also shows that $\mathcal{P}$ will not output an incorrect prediction. It remains to show that $\mathcal{P}$ did not query to h the value X_{i-1} for any $i \in B$. To prove this, we first show that X_i is placed into the table no later than step β_i of $\mathcal{P}$, by induction on β_i. The base case is X_0, which is in the table at step 0. If β_i is an integer, then $i \in B$ and this is true because of step β_i of $\mathcal{P}$. If β_i is not an integer, then $\beta_{i-1} < \beta_i$ (because X_{i-1} appears as a query at round $\lfloor \beta_i \rfloor$), so at the beginning of step β_i of $\mathcal{P}$, by the inductive hypothesis, position $i-1$ in the table will already contain X_{i-1}, and thus position i will get filed in when X_{i-1} gets queried to h.)

Note also that X_i cannot be placed into the table earlier than step β_i, so it is placed in the table exactly at step β_i (as long as $\beta_i \neq \infty$, in which case it is placed into the table at the end, when $\mathcal{P}$ fills in the missing positions).

Now suppose, for purposes of contradiction, that $\mathcal{P}$ queries h for some value X_{i-1} for some $i \in B$. That can happen only if at the end of some round k, X_{i-1} is queried by A as part of the output state, but either X_{i-1} or X_i are not in the table at that time.

- If X_{i-1} is not in the table at the beginning of step $k + 1/2$ of $\mathcal{P}$, then $\beta_{i-1} \geq k + 1/2$; but since X_{i-1} is being queried at the end of round k, $\beta_{i-1} \leq k$, which is a contradiction.
- If X_i is not in the table at the beginning of step $k+1/2$ of $\mathcal{P}$, then $\beta_i \geq k+1$ (because β_i is an integer); but since X_{i-1} appears as query in the output state of round k, $\beta_i \leq k + 1/2$, which is also a contradiction.

Thus, $\mathcal{P}$ always achieves its goal.

For the second sentence of the claim, observe that we can simply give $\mathcal{P}$ the hint for the p blue labels with the smallest β values. □

In the next claim, we show that for every input to $\mathcal{P}$, the algorithm $\mathcal{P}$ cannot be correct for too many oracles h.

Claim 2 *Fix an algorithm $\mathcal{P}$ and fix its input, a positive integer p, some domain $\mathcal{D}$, and range $\mathcal{R}$. For $\mathsf{h} : \mathcal{D} \to \mathcal{R}$, call $\mathcal{P}^{\mathsf{h}}$* successful *if $\mathcal{P}$ with oracle access to h outputs p distinct values $x_1, \ldots, x_p \in \mathcal{D}$ and $\mathsf{h}(x_1), \ldots, \mathsf{h}(x_p)$ without querying h on any of $x_1, \ldots, x_p$. Then $\Pr_{\mathsf{h} \in \mathcal{R}^{\mathcal{D}}}[\mathcal{P}^{\mathsf{h}}$ is successful$] \leq |\mathcal{R}|^{-p}$.*

Proof. Instead of choosing h all at once, consider the equivalent view of choosing answers to fresh queries of $\mathcal{P}$ uniformly at random, and then choosing the remainder of the h uniformly at random after $\mathcal{P}$ produces its output. Since $\mathcal{P}$ does not query $x_1, \ldots, x_p$, the choices of h on those points will agree with $y_1, \ldots, y_p$ with the probability at most $|\mathcal{R}|^{-p}$. □

Using the previous two claims, we can now bound the number of random oracles for which the size of the blue set is too large. Recall that B is the blue set minus X_0.

Claim 3 *Given adversary A, there exists a set of random oracles $\mathsf{good}_{\mathsf{h}}$ such that $\Pr[\mathsf{h} \notin \mathsf{good}_{\mathsf{h}}] \leq qn^3 2^{-w}$, and for every $\mathsf{h} \in \mathsf{good}_{\mathsf{h}}$, every M, and every initial state σ_0 of A of size at most M bits, $|B| \leq p - 1$, where $p = \lceil (M + 1)/(w - 2\log n - \log q) + 1 \rceil$.*

Proof. The intuition is as follows: if for some h and some initial input state of length M, $|B| > p - 1$, then either $\mathcal{P}$ successfully predicts the output of h on p distinct inputs (by Claim 1), or some of the values among $X_0, \ldots, X_{n-1}$ are not distinct. We will define $\mathsf{bad}_{\mathsf{h}}$ as the set of random oracles for which this can happen, and then bound its size.

Let $\mathcal{S}$ be the size of the space of all possible random oracles h. There are at most $\frac{1}{2}\mathcal{S}n^2 2^{-w}$ random oracles for which some of the values among $X_0, \ldots, X_{n-1}$ are not distinct; (suppose the first collision pair is $i, j < n$, thus $X_{i-1} \neq X_{j-1}$, and the probability that $X_i = X_j$ is 2^{-w}; then the bound is given by taking union bound over at most $n^2/2$ pairs of (i, j).) Call this set of random oracles $\mathsf{colliding}$.

In the next paragraph, we will formally define the set $\mathsf{predictable}$ as the set of random oracles for which $\mathcal{P}$ correctly predicts the output on p distinct inputs given the M-bit input state of A and an additional $p(2\log n + \log q)$-bit hint. We will bound the size of $\mathsf{predictable}$ by bounding it for every possible memory state of A and every possible hint, and then taking the union bound over all memory states and hints.

Consider a particular input state of length M for A; recall that $p = \lceil (M + 1)/(w - 2\log n - \log q) + 1 \rceil$. Assume $1 \leq p \leq n - 1$ (otherwise, the statement of Claim 3 is trivially true). Fix a particular value of the hint for $\mathcal{P}$ for predicting p elements of B. (Recall that the hint was previously defined dependent on the random oracle; we are now switching the order of events by fixing the hint first and then seeing for how many random oracles this hint can work.) Since the input to $\mathcal{P}$ is now fixed, there are most $\mathcal{S}2^{-pw}$ random oracles for which it can correctly output p distinct values without querying them, by Claim 2. The set $\mathsf{predictable}$ consists of all such random oracles, for every value of M such that $p \leq n$, every M-bit input state σ_0, and every hint.

To count how many random oracles are in $\mathsf{predictable}$, first fix p. Let M_p be the largest input state length that gives this particular p. Take all input state lengths that give this p, all possible input states of those lengths (there are at most $2^{M_p} + 2^{M_p - 1} + \cdots + 1 < 2^{M_p+1}$ of them), and all possible hints for extracting p values (there are at most $2^{p(2\log n + \log q)}$ of them). This gives us at most $\mathcal{S}2^{(M_p+1)+p(2\log n+\log q - w)}$ random oracles in $\mathsf{predictable}$. Since $(M_p + 1) \leq (p-1)(w - 2\log n - \log q)$ by definition of p, this number does not exceed $\mathcal{S}2^{(2\log n + \log q - w)} = \mathcal{S}n^2 q 2^{-w}$. Now add up over all possible values of p (from 2 to n), to get $|\mathsf{predictable}| \leq \mathcal{S}(n-1)n^2 q 2^{-w}$.

Set $\mathsf{bad_h} = \mathsf{colliding} \cup \mathsf{predictable}$ and let $\mathsf{good_h}$ be the complement of $\mathsf{bad_h}$.

□

Claim 4 *For every i, $0 \leq i < n$, the value t_i is at least $1 + i - j$, where $j = max\{a \leq i \,|\, a \text{ is blue}\}$.*

Proof. If i is blue, we are done, since $t_i \geq 1$ simply because we start counting rounds from 1.

We will first show that $\lceil \beta_i - \beta_j \rceil \geq i - j$. Fix a blue j and proceed by induction on i such that $i > j$ and there are no blue indices greater than j and less than i. For the base case, suppose $i = j + 1$. Recall that β_i is not an integer because i is not blue. Then $\beta_{i-1} \leq \beta_i - 1/2$, because X_{i-1} is present as the query to h that produces response X_i in the sequence of queries that A makes when responding to the challenge $\mathsf{bestchal}_i$, and we are done. For the inductive case, it suffices to show that $\beta_{i-1} \leq \beta_i - 1$, which is true by the same argument as for the base case, except that we add that β_{i-1} is also not an integer (since $i-1$ is also not blue).

Therefore, $\lceil \beta_i \rceil \geq i - j + 1$, because $\beta_j \geq 1$. We thus have $\pi_{ii} = t_i \geq \lceil \beta_i \rceil \geq i - j + 1$. □

The number of blue indices (namely, $|B|+1$, because X_0 is blue but not in B) is at most p if $h \in \mathsf{good_h}$. Since at most d indices are within distance $d-1$ of any given blue index, and there are at most p blue indices, we can plug in $d = n(1 - \mathrm{pr}_{\mathrm{hard}})/p$ to get

$$\Pr_i \left[t_i \leq \frac{n(1 - \mathrm{pr}_{\mathrm{hard}})}{p} \right] \leq 1 - \mathrm{pr}_{\mathrm{hard}}.$$

This concludes the proof of Theorem 2. □

Generalizing to other graphs. In general, every DAG G defines a (data-independent) function whose evaluations on input X corresponds to labeling G as follows: The source is labeled with X, and the label of every node is obtained by hashing the concatenation of the labels of its predecessors. Rather than evaluating this function, one can instead consider a game with challenges, where in each round, the adversary needs to compute the label of a random challenge node from G. Theorem 2 above can be seen as dealing with the special case where G is a line graph.

However, we note that the proof of Theorem 2 is independent of the specific graph until Claim 4 (the only difference is that the random oracle query resulting in response X_i is not necessarily X_{i-1}, but perhaps a concatenation of other labels, and there may be more source nodes than just X_0). Claim 4 can be generalized as follows: for every i, the amount of time required to query X_i is at least one plus the longest path that starts at a blue node, goes through no other blue nodes, and ends at X_i. The proof is of this more general claim is the same. With this more general claim in place, it follows

$$\Pr_i [t_i \geq m] \geq \frac{1}{2},$$

where $m = \min_{\text{set } B \text{ of } p-1 \text{ nodes}}$ (median, over all $v \in G$, longest path that starts at B or at a source node and ends in v without going through other nodes in B). Of course, other statistical properties of the distribution of t_i can also be deduced from this claim. Essentially, this shows that the best the adversary can do is place $p-1$ pebbles on the graph and use parallel pebbling.

5 Cumulative Complexity of Answering Repeated Challenges in the Parallel Pebbling Model

In the previous part, we showed that, in the parallel random oracle model, an adversary with memory (input state) of size M cannot do much better in answering a random challenge than placing $\mathsf{p} \approx M/w$ pebbles on the graph and pebbling. In this section, we prove a lower bound on the cumulative complexity of *repeated* random challenges in the pebbling model. While the result in this section does not directly apply to the random oracle model for reasons explained in Section 5.1, all of the techniques are used in the proof of our main theorem in Section 6.

The Single Challenge Pebbling Game. Consider now the pebbling game for the line graph G consisting of nodes $v_0, \ldots, v_{n-1}$ and edges (v_i, v_{i+1}) for every $0 \le i < n$. Recall that in this game, at each time step t starting with $t = 0$, the adversary maintains a subset P_t of nodes ("pebbles"). If there is a pebble on a node at time $t-1$, its successor is allowed (but not required) to get a pebble at time t. Formally, at time $t > 0$, the set P_t must be a subset of $P_{t-1} \cup \{v_{i+1} \,|\, v_i \in P_{t-1}\}$. Also recall that we modify the game of [AS15] slightly by not permitting the adversary to put a pebble on the source for free: v_0 is contained in P_0 and cannot be added to P_t if it is absent in P_{t-1} (this change simplifies calculations). Let $\mathsf{p}_i = |P_i| \ge 1$.

We will say that the adversary answers a challenge chal (for $0 \le \mathsf{chal} < n$) in t steps if $t > 0$ is the earliest time when $v_{\mathsf{chal}} \in P_{t-1}$ (note that a pebble needs to be on v_{chal} at time $t-1$—think of time t as the step when the output to the challenge is presented; this convention again simplifies calculations, and intuitively corresponds to the `scrypt` evaluation, in which the "output" step corresponds to querying $X_{\mathsf{chal}} \oplus S_i$ in order to advance to the next challenge).

It is easy to see that t is at least one plus the distance between chal and the nearest predecessor of chal in P_0. Therefore, for the same reason as in the proof of Theorem 2 (because at most $n/(2\mathsf{p}_0)$ challenges are within $n/(2\mathsf{p}_0) - 1$ distance to a particular node in P_0 and there are p_0 nodes in P_0).

$$\Pr_{\mathsf{chal}}\left[t > \frac{n}{2\mathsf{p}_0}\right] \ge \frac{1}{2}\,.$$

More generally, the following is true for any $0 \le \mathrm{pr}_{\mathrm{hard}} \le 1$ and $c = n(1 - \mathrm{pr}_{\mathrm{hard}})$:

Fact 3

$$\Pr_{\mathsf{chal}}\left[t > \frac{c}{\mathsf{p}_0}\right] \ge \mathrm{pr}_{\mathrm{hard}}\,.$$

Repeated Challenges Pebbling Game. We now consider repeated challenges. At time $s_1 = 0$, the adversary receives a challenge c_1, $0 \le c_1 < n$. The adversary answers this challenge at the earliest moment $s_2 > s_1$ when P_{s_2-1} contains X_{c_1}; after P_{s_2} is determined, the adversary receives the next challenge c_2, and so on, for Q challenges, until challenge c_Q is answered at time s_{Q+1}. We are interested in the cumulative pebbling complexity $\mathsf{cc}_{\mathsf{peb}} = \sum_{t=0}^{s_{Q+1}} \mathsf{p}_t$.

Note that the adversary can adaptively vary the number of pebbles used throughout the game, while Fact 3 above addresses only the number of pebbles used before a challenge is known. Nevertheless, we are able to show the following result.

Theorem 4 (Cumulative pebbling complexity of repeated challenges game). *The cumulative pebbling complexity repeated challenges pebbling game is with high probability $\Omega(nQ)$.*

More precisely, suppose the adversary never has fewer than p_0 pebbles. Then for any $\epsilon > 0$, with probability at least $1 - e^{-2\epsilon^2 Q}$ over the choice of the Q challenges,

$$\mathsf{cc}_{\mathsf{peb}} \geq \mathsf{p}_0 + \frac{n}{2} \cdot Q \cdot \left(\frac{1}{2} - \epsilon\right) \cdot \ln 2\,.$$

More generally, we replace the condition that the adversary never has fewer than p_0 pebbles with the condition $\mathsf{p}_t \geq \mathsf{p}_{\min}$ for some $\mathsf{p}_{\min}$ and every $t \geq 1$, we need to replace $\ln 2$ with

$$\ln\left(1 + \left(\frac{\mathsf{p}_{\min}}{\mathsf{p}_0}\right)^{\frac{1}{Q\left(\frac{1}{2}-\epsilon\right)}}\right)\,.$$

This result improves [ACK+16b, Theorem 1] by eliminating the $\log^2 n$ factor from cumulative memory complexity of pebbling game. In Appendix B we discuss the general case of $\mathsf{p}_t \geq \mathsf{p}_{\min}$ and show an attack showing that a bound as the above is necessary (up to constant factors in the exponent).

Our approach is general enough to apply to time-space tradeoffs other than inverse proportionality, other graphs, and even some other models of computation that do not deal with pebbling. However, we will explain in Section 5.1 why it cannot be used without modification in the parallel random oracle model and other models where space is measured in bits of memory.

Proof. Recall time starts at 0, p_t denotes the number of pebbles at time t, and s_i denotes the moment in time when challenge number i (with $1 \leq i \leq Q$) is issued. Let t_i denote the amount of time needed to answer challenge number i (thus, $s_1 = 0$ and $s_{i+1} = s_i + t_i$; let $s_{Q+1} = s_Q + t_Q$). Let $\mathsf{cc}(t_1, t_2)$ denote $\sum_{t=t_1}^{t_2} \mathsf{p}_t$.

The main idea of the proof The difficulty in the proof is that we cannot use t_i to infer anything about the number of pebbles used during each step of answering challenge i. All we know is that number of pebbles has to be inversely proportional to t_i immediately before the challenge was issued—but the adversary can then reduce the number of pebbles used once the challenge is known (for, example by keeping pebbles only on v_0 and on the predecessor of the challenge).

The trick to overcome this difficulty is to consider how many pebbles the adversary has to have in order to answer the next challenge not only immediately before the challenge, but one step, two steps, three steps, etc., earlier.

Warm-up: starting with a stronger assumption For a warm-up, consider the case when the pebbles/time tradeoff is guaranteed (rather than probabilistic, as in Fact 3): assume, for now, that in order to answer the next random challenge in time t, it is necessary to have a state of size c/t right before the challenge is issued. Now apply this stronger assumption not only to the moment s in time when the challenge is issued, but also to a moment in time some j steps earlier.

The assumption implies that the number of pebbles needed at time $s-j$ is at least $c/(j+t)$ (because the challenge was answered in $j+t$ steps starting from time $s-j$, which would be impossible with a lower number of pebbles even if the challenge had been already known at time $s-j$).

We will use this bound for every challenge number $i \geq 2$, and for every $j=0$ to t_{i-1}, i.e., during the entire time the previous challenge is being answered. Thus, cumulative pebbling complexity during the time period of answering challenge $i-1$ is at least

$$\begin{aligned}\mathsf{cc}(s_{i-1}+1, s_i) &\geq \sum_{j=0}^{t_{i-1}-1} \mathsf{p}_{s_i-j} \geq c\left(\frac{1}{t_i}+\frac{1}{t_i+1}+\cdots+\frac{1}{t_i+t_{i-1}-1}\right)\\ &\geq c\int_{t_i}^{t_{i-1}+t_i}\frac{dx}{x} = c(\ln(t_{i-1}+t_i)-\ln t_i)\,.\end{aligned}$$

Then adding these up for each i between 2 and Q, we get the cumulative pebbling complexity of

$$\mathsf{cc}(1, s_{Q+1}) \geq c\sum_{i=2}^{Q}(\ln(t_{i-1}+t_i)-\ln t_i)\,.$$

If all t_i are equal (which is close to the minimum, as we will show below), this becomes $c(Q-1)\cdot\ln 2$.

Back to the actual assumption The proof is made messier by the fact that the bound in the assumption is not absolute. Moreover, the bound does not give the number of pebbles in terms of running time, but rather running time in terms of the number of pebbles (it makes no sense to talk probabilistically of the number of pebbles, because the number of pebbles is determined by the adversary before the challenge is chosen). To overcome this problem, we look at the number of pebbles at all times before s_i and see which one gives us the best bound on t_i.

Specifically, suppose the adversary has already answered the first $i-1$ challenges. By Fact 3 applied to any time $t \leq s_i$, with probability at least $\mathrm{pr}_{\mathrm{hard}}$ over the choice of the ith challenge, $t_i + (s_i - t) > c/\mathsf{p}_t$, i.e., $t_i > c/\mathsf{p}_t - (s_i - t)$. Let r_i be a moment in time that gives the best bound on t_i:

$$r_i = \operatorname*{argmax}_{0\leq t\leq s_i}\left(\frac{c}{\mathsf{p}_t}-(s_i-t)\right)\,.$$

Call the ith challenge "hard" if $t_i + (s_i - r_i) > c/\mathsf{p}_{r_i}$. By Fact 3, each challenge is hard with probability at least $\mathrm{pr}_{\mathrm{hard}}$, and, because the challenges are independent, we can apply Hoeffding's inequality [Hoe63] to obtain that with probability at least $1-e^{-2\epsilon^2 Q}$, the number of hard challenges is at least $h \geq Q(\mathrm{pr}_{\mathrm{hard}}-\epsilon)$.

We claim that if challenge i is hard, then the same fact about number of pebbles j steps before the challenge as we used in the warm-up proof holds.

Claim 5 *If challenge i is hard, then for any j, $0 \leq j \leq s_i$, $\mathsf{p}_{s_i-j} > c/(t_i+j)$.*

Proof. Indeed, let $t = s_i - j$. Then $c/\mathsf{p}_{s_i-j} - j = c/\mathsf{p}_t - (s_i - t) \le c/\mathsf{p}_{r_i} - (s_i - r_i)$ by the choice of r_i. This value is less than t_i by definition of a hard challenge. Therefore, $c/\mathsf{p}_{s_i-j} - j < t_i$ and the result is obtained by rearranging the terms. □

What remains to show is a purely algebraic statement about the sum of p_i values when $h \ge Q(\mathrm{pr}_{\mathrm{hard}} - \epsilon)$ of challenges satisfy Claim 5.

Claim 6 *Let c be a real value. Let $t_1, \dots, t_Q$ be integers, $s_1 = 0$, and $s_i = s_{i-1} + t_i$ for $i = 2, \dots, Q+1$. Let $\mathsf{p}_0, \dots, \mathsf{p}_Q$ be a sequence of real values with $\mathsf{p}_t > \mathsf{p}_{\min}$ for every $t \ge 1$. Suppose further that there exist at least h distinct indices i, with $1 \le i \le Q$ (called "hard indices") such that for any $0 \le j \le s_i$, $\mathsf{p}_{s_i-j} \ge c/(t_i + j)$. Then*

$$\sum_{i=1}^{s_{Q+1}} \mathsf{p}_i \ge c \cdot h \cdot \ln\left(1 + \left(\frac{\mathsf{p}_{\min}}{\mathsf{p}_0}\right)^{\frac{1}{h}}\right)$$

Proof. Let $i_1 < i_2 < \cdots < i_h$ be the hard indices. Recall notation $\mathsf{cc}(i,j) = \sum_{t=i}^{j} \mathsf{p}_t$. Then for $k \ge 2$,

$$\begin{aligned}\mathsf{cc}(s_{i_{k-1}} + 1, s_{i_k}) &\ge \mathsf{cc}(s_{i_k} - t_{i_{k-1}} + 1, s_{i_k})\\ &= \sum_{j=0}^{t_{i_{k-1}}-1} \mathsf{p}_{s_{i_k}-j}\\ &\ge \sum_{j=0}^{t_{i_{k-1}}-1} \frac{c}{t_{i_k} + j} \ge c \cdot (\ln(t_{i_{k-1}} + t_{i_k}) - \ln t_{i_k})\end{aligned}$$

(the last inequality follows by the same reasoning as in the warm-up). To bound $\mathsf{cc}(1, Q+1)$, we will add up the pebbling complexity during these nonoverlapping time periods for each k and find the minimum over all sets of values of t_{i_k}. Unfortunately, the result will decrease as t_{i_1} decreases and as t_{i_h} increases, and we have no bounds on these values. To get a better result, we will need to consider special cases of $k = 2$ (to replace t_{i_1} with c/p_0) and $k = h + 1$ (to add another term with $t_{i_h+1} = c/\mathsf{p}_{\min}$).

For $k = 2$, we will bound $\mathsf{cc}(1, s_{i_2})$ by noticing that $s_{i_2} \ge t_{i_1} + s_{i_1} \ge c/\mathsf{p}_0$ (where the second step follows by Claim 5 with $j = s_{i_1}$), and therefore

$$\begin{aligned}\mathsf{cc}(1, s_{i_2}) &\ge \sum_{j=0}^{s_{i_2}-1} \mathsf{p}_{s_{i_2}-j}\\ &\ge c \sum_{j=0}^{s_{i_2}-1} \frac{1}{t_{i_2} + j} \ge c \int_{t_{i_2}}^{s_{i_2}+t_{i_2}} \frac{dx}{x}\\ &= c(\ln(s_{i_2} + t_{i_2}) - \ln t_{i_2}) \ge c \cdot (\ln(c/\mathsf{p}_0 + t_{i_2}) - \ln t_{i_2}) .\end{aligned}$$

For $k = h+1$,

$$\begin{aligned}\mathsf{cc}(s_{i_h}+1, s_{h+1}) \geq \mathsf{p}_{\min}\cdot t_{i_h} &\geq c\cdot\left(\frac{1}{c/\mathsf{p}_{\min}} t_{i_h}\right)\\ &\geq c\cdot\left(\frac{1}{c/\mathsf{p}_{\min}} + \cdots + \frac{1}{c/\mathsf{p}_{\min}+t_{i_h}-1}\right)\\ &\geq c\cdot\left(\ln(t_{i_h}+c/\mathsf{p}_{\min}) - \ln c/\mathsf{p}_{\min}\right).\end{aligned}$$

Adding these up, we get

$$\begin{aligned}\mathsf{cc}(0, s_{i+1}) = \mathsf{p}_0 + \mathsf{cc}(1, s_{i_2}) + \mathsf{cc}(s_{i_2}+1, s_{i_3}) + \ldots\\ + \mathsf{cc}(s_{i_{h-1}}+1, s_{i_h}) + \mathsf{cc}(s_{i_h}+1, s_{i_{h+1}})\\ \geq \mathsf{p}_0 + c\cdot\sum_{i=1}^{h}(\ln(x_i+x_{i+1}) - \ln x_{i+1}),\end{aligned}$$

where $x_1 = c/\mathsf{p}_0$, $x_2 = t_{i_2}$, $x_3 = t_{i_3}$, …, $x_h = t_{i_h}$, and $x_{h+1} = c/\mathsf{p}_{\min}$.

To find the minimum of this function, observe that the first derivative with respect to x_i is $\frac{1}{x_i+x_{i-1}} + \frac{1}{x_i+x_{i+1}} - \frac{1}{x_i}$, which, assuming all the x_is are positive, is zero at $x_i = \sqrt{x_{i-1}x_{i+1}}$, is negative for $x_i < \sqrt{x_{i-1}x_{i+1}}$, and is positive for $x_i > \sqrt{x_{i-1}x_{i+1}}$. Therefore, the minimum of this function occurs when each x_i, for $2 \leq i \leq h$, is equal to $\sqrt{x_{i-1}x_{i+1}}$, or equivalently, when $x_i = c/(\mathsf{p}_{\min}^{i-1}\mathsf{p}_0^{h-i+1})^{1/h}$. This setting of x_i gives us $\ln(x_i+x_{i+1}) - \ln x_{i+1} = \ln(1+(\mathsf{p}_{\min}/\mathsf{p}_0)^{1/h})$, which gives the desired result. □

Plugging in $Q\cdot(\mathrm{pr}_{\mathrm{hard}}-\epsilon)$ for h and $\mathrm{pr}_{\mathrm{hard}} = \frac{1}{2}$ concludes the proof of Theorem 4. □

5.1 Why this proof needs to be modified for the parallel random oracle model

The main idea of the proof above is to apply the space-time tradeoff of Fact 3 to every point in time before the challenge is known, arguing that delaying the receipt of the challenge can only hurt the adversary. While this is true in the pebbling game (via an easy formal reduction—because getting bits does not help get pebbles), it's not clear why this should be true in the parallel random oracle game, where space is measured in bits rather than pebbles. A reduction from the adversary A who does not know the challenge to an adversary B who does would require B to store the challenge in memory, run A until the right moment in time, and then give A the challenge. This reduction consumes memory of B—for storing the challenge and keeping track of time. In other words, A can save on memory by not knowing, and therefore not storing, the challenge. While the amount of memory is small, it has to be accounted for, which makes the formulas even messier. Things get even messier if A is not required to be 100% correct, because, depending on the exact definition of the round game, B may not know when to issue the challenge to A.

Nevertheless, this difficulty can be overcome when challenges come from the random oracle, as they do in `scrypt`. We do so in the next section.

6 Main Result: Memory Hardness of scrypt in the Parallel Random Oracle Model

We are ready to restate our main theorem, which we now prove extending the techniques from the two previous sections. We state it with a bit more detail than Theorem 1.

Fix positive integers $n \geq 2$ and w, a string $X \in \{0,1\}^w$, a finite domain $\mathcal{D}$ that contains at least $\{0,1\}^w$, and let $\mathcal{R} = \{0,1\}^w$.

Theorem 5. *Let A be any oracle machine (in the parallel random oracle model as defined in Section 2) with input X. Assume $A^{\mathsf{h}}(X)$ outputs $S_n^{\mathsf{h}} = \mathtt{scrypt}^{\mathsf{h}}(X)$ correctly with probability χ, where the probability is taken over the choice of $\mathsf{h} : \mathcal{D} \to \mathcal{R}$. Then for any $\epsilon > 0$ and $q \geq 2$, with probability (over the choice h) at least $\chi - 2qn^4 \cdot 2^{-w} - e^{-2\epsilon^2 n}$ one of the following two statements holds: either $A^{\mathsf{h}}(X)$ makes more than q queries (and thus $\mathsf{cc}_{\mathsf{mem}}(A^{\mathsf{h}_n}) > qw$ by definition) or*

$$\mathsf{cc}_{\mathsf{mem}}(A^{\mathsf{h}}(X)) \geq \frac{\ln 2}{6} \cdot \left(\frac{1}{2} - \epsilon\right) \cdot n^2 \cdot (w - 2\log n - \log q - 1) \ .$$

To get the statement of Theorem 1, we set $\epsilon = 1/7$ and observe that then $e^{-2\epsilon^2 n} = 2^{-\frac{2n}{49 \ln 2}} < 2^{-n/20}$ and $\frac{\ln 2}{6}(\frac{1}{2} - \frac{1}{7}) > \frac{1}{25}$. We also plug in $q = \min(2, \frac{n^2}{25})$ (and therefore $\log q \leq 2\log n - 1$), thus removing the "either/or" clause (this setting of q requires us to manually check that the probability statement is correct when $q > \frac{n^2}{25}$, i.e., $2 \leq n \leq 7$—a tedious process that we omit here).

The rest of this section is devoted to the proof of this theorem.

6.1 Outline of the Approach

Before proceeding with the proof, we justify our proof strategy by highlighting the challenges of extending Theorem 2 and Theorem 4 to this setting. Theorem 2 applies to a fixed random oracle h and a random challenge. In fact, the proof relies crucially on the ability to try every challenge for a given oracle. However, in the present proof, once the random oracle is fixed, so is every challenge. Moreover, Theorem 4 crucially relies on the uniformity and independence of each challenge, which is issued only when the previous challenge is answered. In contrast, here, again, once the oracle is fixed, the challenges are fixed, as well. Even if we think of the oracle as being lazily created in response to queries, the challenges implicitly contained in the answers to these queries are not necessarily independent once we condition (as we need to in Theorem 2) on the oracle not being in $\mathsf{bad}_{\mathsf{h}}$. We resolve these issues by working with multiple carefully chosen random oracles.

Recall our notation: $X_0 = X$, $X_1 = \mathsf{h}(X_0), \ldots, X_{n-1} = \mathsf{h}(X_{n-2})$; $T_0 = X_{n-1}$, $S_0 = \mathsf{h}(T_0)$, and for $i = 1, \ldots, n$, $T_i = S_{i-1} \oplus X_{S_{i-1} \bmod n}$ and $S_i = \mathsf{h}(T_i)$. Because we will need to speak of different random oracles, we will use notation X_i^{h}, T_i^{h}, and S_i^{h} when the label values are being computed with respect to the random oracle h, unless the specific instance of the random oracle is clear from the context. We will denote by A^{h} the adversary running with oracle h. (To simplify notation, we will omit the argument X to the adversary A for the remainder of this section, since it is fixed.)

Let changeModn(S, i) be a function that keeps the quotient $\lfloor S/n \rfloor$ but changes the remainder of S modulo n to i. Consider the following process of choosing a random oracle (this process is described more precisely in the following section). Choose uniformly at random an oracle h_0. Choose uniformly at random challenges $c_1, \ldots, c_n$, each between 0 and $n-1$. Let h_1 be equal to h_0 at every point, except $\mathsf{h}_1(T_0^{\mathsf{h}_0}) = \text{changeModn}(S_0^{\mathsf{h}_0}, c_1)$. Similarly, let h_2 be equal to h_1 at every point, except $\mathsf{h}_2(T_1^{\mathsf{h}_1}) = \text{changeModn}(S_1^{\mathsf{h}_1}, c_2)$, and so on, until h_n, which is our final random oracle.

This method of choosing h_n is close to uniform, and yet explicitly embeds a uniform random challenge. Unless some (rare) bad choices have been made, each challenge has about a $\frac{1}{2}$ probability of taking a long time to answer, by the same reasoning as in Theorem 2. And since the challenges are independent (explicitly through the choices of c_i values), we can use the same reasoning as in Theorem 4 to bound the cumulative complexity.

The main technical difficulty that remains is to define exactly what those bad choices are and bound their probability without affecting the independence of the challenges. In particular, there are n^n possible challenge combinations, and the probability that all of them yield random oracles that are acceptable (cause no collisions and cannot be predicted) is not high enough. We have to proceed more carefully.

The first insight is that if h_{k-1} is not predictable (i.e., a predictor $\mathcal{P}$ with a short input cannot correctly extract many oracle values), and no oracle queries up to T_{k-1} collide, then $\Pr_{c_k}[\text{time between queries } T_{k-1}^{\mathsf{h}_k} \text{ and } T_k^{\mathsf{h}_k} \text{ is high}] \geq \frac{1}{2}$, by the same reasoning as in Theorem 2 (except predictor needs an extra $\log q$ bits of hint to know when query T_{k-1} occurs, so as to substitute the answer to T_{k-1} with changeModn(S_{k-1}, c_k) for every possible c_k). This allows us to worry about only n random oracles avoiding collisions and the set $\mathsf{predictable}$ (instead of worrying about n^n random oracles) to ensure that the time between consecutive challenges is likely to be high.

However, the reasoning in the previous paragraph bounds the time required to answer the challenge c_k only with respect to oracle h_k. In order to reason about A interacting with oracle h_n, we observe that if for every k, A_k^{h} asks the queries $X_0, \ldots, X_{n-1} = T_0, T_1, \ldots, T_n$ in the correct order, then the computation of A^{h_n} is the same as the computation of A^{h_k} until the kth challenge is answered—i.e., until T_k is queried. Thus, results about each of the oracles h_k apply to h_n.

The rest of the work involves a careful probability analysis to argue that the challenges $c_1, \ldots, c_n$ are almost independent even when conditioned on the all the bad events not happen, and to bound the probability of these events.

6.2 The Detailed Proof

Recall that we assume that the adversary A is deterministic without loss of generality (this fact will be used heavily throughout the proof). In particular, the randomness of the experiment consists solely of the random oracle A is given access to.

Following up on the above high-level overview, we now make precise the definition of h_k. Let h_0 be a uniformly chosen random oracle. Let changeModn(S, i) be a function that keeps the quotient $\lfloor S/n \rfloor$ but changes the remainder of S modulo n to i if possible: it views S as an integer in $[0, 2^w - 1]$, computes $S' = \lfloor S/n \rfloor \cdot n + i$, and outputs S' (viewed as a w-bit string) if $S' < 2^w$, and S otherwise (which can happen only if n is not a power of 2, and even then is very unlikely for a random S).

Definition 1. *Let* $\mathsf{roundingProblem}_k$ *be the set of all random oracles* h *for which the value of at least one of* $S_0^{\mathsf{h}}, \dots, S_k^{\mathsf{h}}$ *is greater than* $\lfloor 2^w/n \rfloor \cdot n - 1$ *(i.e., those for which* changeModn *does not work on some* S *value up* S_k*).*

Definition 2. *Let* $\mathsf{colliding}_k^*$ *be the set of all* h *which there is at least one collision among the values* $\{X_0, X_1^{\mathsf{h}}, X_2^{\mathsf{h}}, \dots, X_{n-2}^{\mathsf{h}}, T_0^{\mathsf{h}}, T_1^{\mathsf{h}}, \dots, T_k^{\mathsf{h}}\}$*. Let* $\mathsf{colliding}_k = \mathsf{roundingProblem}_{\min(k,n-1)} \cup \mathsf{colliding}_k^*$.

Definition 3. *For every* k $(0 \leq k < n)$*, let* $\mathsf{h}_{k+1} = \mathsf{h}_k$ *if* $\mathsf{h}_k \in \mathsf{colliding}_k$*; else, choose* c_{k+1} *uniformly at random between* 0 *and* $n-1$*, let* $\mathsf{h}_{k+1}(T_k^{\mathsf{h}_k}) =$ changeModn$(S_k^{\mathsf{h}_k}, c_{k+1})$*, and let* $\mathsf{h}_{k+1}(x) = \mathsf{h}_k(x)$ *for every* $x \neq T_k^{\mathsf{h}_k}$*. (Recall that* h_0 *is chosen uniformly.)*

Note that this particular way of choosing h_{k+1} is designed to ensure that it is uniform, as we will argue in Claim 11.

THE SINGLE CHALLENGE ARGUMENT. In the argument in Theorem 2, the predictor issues different challenges to A. Here, instead, the predictor will run A with different oracles. Specifically, given $1 \leq k \leq n$ and a particular oracle $\mathsf{h}_{k-1} \notin \mathsf{colliding}_{k-1}$, consider the n oracles $\mathsf{h}_{k,j}$ for each $0 \leq j < n$, defined to be the same as h_{k-1}, except $\mathsf{h}_{k,j}(T_{k-1}^{\mathsf{h}_{k-1}}) =$ changeModn$(S_{k-1}^{\mathsf{h}_{k-1}}, j)$ (instead of $S_{k-1}^{\mathsf{h}_{k-1}}$).

Since $\mathsf{h}_{k-1} \notin \mathsf{colliding}_{k-1}$, $T_{k-1}^{\mathsf{h}_{k-1}}$ is not equal to $X_i^{\mathsf{h}_{k-1}}$ for any $0 \leq i < n-1$ and $T_i^{\mathsf{h}_{k-1}}$ for any $0 \leq i < k-1$. Therefore (since h_{k-1} and $\mathsf{h}_{k,j}$ differ only at the point $T_{k-1}^{\mathsf{h}_{k-1}}$), we have $X_i^{\mathsf{h}_{k-1}} = X_i^{\mathsf{h}_{k,j}}$ for every $0 \leq i \leq n-1$ and $T_i^{\mathsf{h}_{k-1}} = T_i^{\mathsf{h}_{k,j}}$ for any $0 \leq i \leq k-1$. In particular, the execution of A with oracle $\mathsf{h}_{k,j}$ will proceed identically for any j (and identically to the execution of $A^{\mathsf{h}_{k-1}}$) up to the point when the query T_{k-1} is first made (if ever). We will therefore omit the superscript on T_{k-1} for the remainder of this argument.

The observation is that the moment T_{k-1} is queried is the moment when the predictor argument of Theorem 2 can work, by having the predictor substitute different answers to this query and run A on these different answers in parallel. However, since Section 5 requires a time/memory tradeoff for every point in time before the challenge is given, we will prove a more general result for any point in time before T_{k-1} is queried.

We number all the oracle queries that A makes across all rounds, sequentially. We will only care about the first q oracle queries that A makes, for some q to be set later (because if q is too large, then $\mathsf{cc}_{\mathsf{mem}}$ of A is automatically high). Note that q here is analogous to $q-1$ in Theorem 2.

Let $s_k > 0$ be the round in which T_{k-1} is first queried, i.e., contained in $\mathbf{q}_{s_k}$. For an integer $r \leq s_k$, consider the output state $\bar{\sigma}_r$ of $A^{\mathsf{h}_{k-1}}$ from round r. Given $\bar{\sigma}_r$, consider n different continuations of that execution, one for each oracle $\mathsf{h}_{k,j}, 0 \leq j < n$. For each of these continuations, we let $t_j > 0$ be the smallest value such that such $r+t_j > s_k$ and the query $T_k^{\mathsf{h}_{k,j}}$ is contained in $\mathbf{q}_{r+t_j}$ (if ever before query number $q+1$; else, set $t_j = \infty$). We can thus define π_{ij}, β_i, and $\mathsf{bestchal}_i$, blue nodes, and the set B the same way as in Theorem 2, by counting the number of rounds after round r (instead of from 0) and substituting, as appropriate "challenge j" with $\mathsf{h}_{k,j}$ and "query X_j" with "query X_j or $T_k^{\mathsf{h}_{k,j}}$" (note that because $\mathsf{h}_{k-1} \notin \mathsf{roundingProblem}_{k-1}$, $S_{k-1}^{\mathsf{h}_{k,j}} \bmod n = j$, and so $T_k^{\mathsf{h}_{k,j}} = X_j \oplus S_{k-1}^{\mathsf{h}_{k,j}}$). (We stop the execution of A after q total queries in these definitions.)

We now show that, similarly to Claim 1, we can design a predictor algorithm $\mathcal{P}$ that predicts every $X_i^{\mathsf{h}_{k-1}}$ in B by interacting with h_{k-1} but not querying it at the predecessors of points in B. The difference is that instead of running $A^{\mathsf{h}_{k-1}}$ on σ_0 and giving A different challenges j, $\mathcal{P}$ will run A with initial input state σ_r, simulating different oracles $\mathsf{h}_{k,j}$ (which differ from h_{k-1} on only one point—namely, the output on input T_{k-1}). $\mathcal{P}$ gets, as input, σ_r and the same hint as in Claim 1. $\mathcal{P}$ also needs an additional hint: an integer between 1 and q indicating the sequential number (across all queries made in round r or later) of the first time query T_{k-1} occurs, in order to know when to reply with $S_{k-1}^{\mathsf{h}_{k,j}} = \text{changeModn}(S_{k-1}^{\mathsf{h}_{k-1}}, j)$ instead of $S_{k-1}^{\mathsf{h}_{k-1}}$ itself. Note that this substitution will require $\mathcal{P}$ to modify the input state σ_{s_k}. If $s_k > r$, then $\mathcal{P}$ will not only be able to answer with $S_{k-1}^{\mathsf{h}_{k,j}}$, but will also see the query T_{k-1} itself as part of the output state $\bar{\sigma}_{s_k}$, and will therefore be able to answer subsequent queries to T_{k-1} consistently. However, if $s_k = r$, then we need to give T_{k-1} to $\mathcal{P}$ to ensure subsequent queries to T_{k-1} are answered consistently. In order to so without lengthening the input of $\mathcal{P}$, we note that in such a case we do not need $S_{k-1}^{\mathsf{h}_{k-1}}$ in σ_r (since $\mathcal{P}$ can obtain it by querying h_{k-1}), and so we can take out $S_{k-1}^{\mathsf{h}_{k-1}}$ and replace it with T_{k-1} ($\mathcal{P}$ will recognize that this happened by looking at the additional hint that contains the query number for T_{k-1} and noticing that it is smaller than the number of queries z_r in round r).

There is one more small modification: if $X_j \in B$ and $\mathsf{bestchal}_j = j$, then in order to correctly predict X_j itself (assuming $X_j \in B$), $\mathcal{P}$ will need one additional bit of hint, indicating whether X_j is first queried by itself or as part of the "next round," i.e., as part of the query $T_k^{\mathsf{h}_{k,j}} = S_{k-1}^{\mathsf{h}_{k,j}} \oplus X_j$ (in which case $\mathcal{P}$ will need to xor the query with $S_{k-1}^{\mathsf{h}_{k,j}}$, which $\mathcal{P}$ knows, having produced it when answering the query T_{k-1}). Finally, note that that $\log q$ bits suffice for the query number of X_i on challenge $\mathsf{bestchal}_i$, because it is not the same query number as T_{k-1}, because $h_{k-1} \notin \mathsf{colliding}_{k-1}$, so there are $q-1$ possibilities plus the possibility of "never".

We thus need to give (in addition to σ_r) $\log q + |B|(1 + 2\log n + \log q)$ bits of hint to $\mathcal{P}$, and $\mathcal{P}$ is guaranteed to be correct as long as $\mathsf{h}_{k-1} \notin \mathsf{colliding}_{k-1}$.

Suppose σ_r has m_r bits. Claim 2 does not change. We modify Claim 3 as follows. We replace p with a function p_r of the memory size m_r, defined as

$$p_r = \lceil (m_r + 1 + \log q)/(w - 2\log n - \log q - 1) + 1 \rceil \tag{1}$$

(note that it is almost the same as the definition of p, but accounts for the longer hint). We now redefine $\mathsf{predictable}$ according to our new definition of $\mathcal{P}$, p_r, and hint length.

Definition 4. *The set* $\mathsf{predictable}$ *consists of all random oracles* h *for which there exists an input state* σ_r *of size* m_r *(such that* $1 \le p_r \le n-1$*) and a hint of length* $\log q + p_r(1 + 2\log n + \log q)$*, given which* $\mathcal{P}$ *can correctly output* p_r *distinct values from among* $X_1^{\mathsf{h}}, \ldots, X_{n-1}^{\mathsf{h}}$ *without querying them.*

Finally, we replace $\mathsf{bad}_{\mathsf{h}}$ with $\mathsf{colliding}_{k-1} \cup \mathsf{predictable}$. As long as $\mathsf{h}_{k-1} \notin \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$, we are guaranteed that $\Pr_j[t_j > n/(2p_r)] \ge 1/2$, like in Theorem 2.

The discussion above gives us the following lemma (analogous to Theorem 2).

Lemma 1. *Fix any any* k *(*$1 \le k \le n$*). Assume* $\mathsf{h}_{k-1} \notin \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$*. Let* $s_k > 0$ *be the smallest value such that* $T_{k-1}^{\mathsf{h}_{k-1}}$ *is among the queries* $\mathbf{q}_{s_k}$ *during the computation of* $A^{\mathsf{h}_{k-1}}$*. Let* $r \le s_k$ *and* m_r *be the bit-length of the input state* σ_r *of* $A^{\mathsf{h}_{k-1}}$ *in round* $r+1$*. Let* $t_{k,j,r} > 0$ *be such that the first time* $T_k^{\mathsf{h}_{k,j}}$ *is queried by* $A^{\mathsf{h}_{k,j}}$ *after round* s_k *is in round* $r + t_{k,j,r}$ *(let* $t_{k,j,r} = \infty$ *if such a query does not occur after round* σ_k *or does not occur among the first* q *queries, or if* $T_{k-1}^{\mathsf{h}_{k-1}}$ *is never queried). Call* j *"hard" for time* r *if* $t_{k,j,r} > n/(2p_r)$*, where* $p_r = \lceil (m_r + 1 + \log q)/(w - 2\log n - \log q - 1) + 1 \rceil$*. We are guaranteed that*

$$\Pr_j [j \textit{ is hard for time } r] \ge \frac{1}{2}\,.$$

Hardness of Challenge c_k. We continue with the assumptions of Lemma 1. In order to get an analogue of Claim 5, we need to define what it means for a challenge to be hard. Consider running A^{h_k}. Let $t_k > 0$ be such that $T_k^{\mathsf{h}_k}$ is queried for the first time in round $s_k + t_k$ (again, letting $t_k = \infty$ if this query does not occur among the first q queries). Find the round $r_k \le s_k$ such that bit-length m_r of the input state σ_r in round $r_k + 1$ gives us the best bound on t_k using the equation of Lemma 1 (i.e., set $r_k = \operatorname{argmax}_{0 \le r \le s_k}(n/(2p_r) - (s_k - r))$, where m_r denotes the size of the state σ_r at the end of round r, and p_r is the function of m_r defined by Equation 1), and define c_k to be "hard" if it is hard for time r_k.

Definition 5. *A challenge* c_k *is hard if for* $r_k = \operatorname{argmax}_{0 \le r \le s_k}(n/(2p_r) - (s_k - r))$ *we have* $t_{k,c_k,r_k} > n/(2p_r)$*, where* $s_k, t_{k,j,r}$ *and* p_r *are as defined in Lemma 1.*

The Multiple Challenges Argument. So far, we considered hardness of c_k during the run of A with the oracle h_k. We now need to address the actual situation, in which A runs with h_n. We need the following claim, which shows that the actual situation is, most of the time, identical. Define $\mathsf{wrongOrder}_k$ as the set of all random oracles h for which the values $\{T_0^{\mathsf{h}}, T_1^{\mathsf{h}}, \ldots, T_k^{\mathsf{h}}\}$ are not queried by A^{h} in the same order as they appear in the correct evaluation of `scrypt` (when we look at first-time queries only, and only up to the first q queries).

Definition 6. $\mathsf{wrongOrder}_k$ *consists of all* h *for which when there exist* i *and* j *such that* $0 \leq i < j \leq k$ *and, in the run of* A^{h}*, query* T_j^{h} *occurs, while query* T_i^{h} *does not occur before query* T_j^{h} *occurs.*

Claim 7 *If for every* k *(*$0 \leq k \leq n$*),* $\mathsf{h}_k \notin \mathsf{colliding}_k \cup \mathsf{wrongOrder}_k$*, then for every* k *and* $i \leq k$*,* $T_i^{\mathsf{h}_n} = T_i^{\mathsf{h}_k}$*, and the execution of* A^{h_n} *is identical to the execution of* A^{h_k} *until the query* T_k *is first made, which (for* $1 \leq k \leq n$*) happens later than the moment when query* $T_{k-1}^{\mathsf{h}_n} = T_{k-1}^{\mathsf{h}_k}$ *is first made.*

Proof. To prove this claim, we will show, by induction, that for every $j \geq k$ and $i \leq k$, $T_i^{\mathsf{h}_k} = T_i^{\mathsf{h}_j}$, and the execution of A^{h_j} is identical to the execution of A^{h_k} until the query T_k is first made.

The base of induction ($j = k$) is simply a tautology.

The inductive step is as follows. Suppose the statement is true for some $j \geq k$. We will show it for $j + 1$. We already established that if $\mathsf{h}_j \notin \mathsf{colliding}_j$, then $T_i^{\mathsf{h}_j} = T_i^{\mathsf{h}_{j+1}}$ for every $i \leq j$, and is therefore equal to $T_i^{\mathsf{h}_k}$ by the inductive hypothesis. Since h_j and h_{j+1} differ only in their answer to the query $T_j^{\mathsf{h}_j} = T_j^{\mathsf{h}_{j+1}}$, the execution of $A^{\mathsf{h}_{j+1}}$ proceeds identically to the execution of A^{h_j} until this query is first made. Since $\mathsf{h}_j \notin \mathsf{wrongOrder}_j$, this moment is no earlier than when the query T_k is made; therefore, until the point the query T_k is first made, the execution of $A^{\mathsf{h}_{j+1}}$ proceeds identically to the execution of A^{h_j} and thus (by the inductive hypothesis) identically to the execution of A^{h_k}.

The last part of the claim follows because $\mathsf{h}_n \notin \mathsf{wrongOrder}_n$. □

We therefore get the following analogue of Claim 5.

Claim 8 *Given adversary* A*, assume for every* k *(*$0 \leq k \leq n$*),* $\mathsf{h}_k \notin \mathsf{colliding}_k \cup \mathsf{wrongOrder}_k$*. If challenge* i *is hard (i.e.,* $t_i + (s_i - r_i) > c/p_{r_i}$*), then, during the run of* A^{h_n}*, for any* $0 \leq j \leq s_i$*,* $p_{s_i - j} \geq c/(t_i + j)$ *for* $c = n/2$*.*

Definition 7. *Let* E_1 *be the event that there are at least* $h \geq n(\frac{1}{2} - \epsilon)$ *hard challenges (as defined in Definition 5). Let* E_2 *be the event that* $\mathsf{h}_k \notin \mathsf{colliding}_k \cup \mathsf{wrongOrder}_k$ *(see Definitions 2 and 6) for every* k*, and* A^{h_n} *queries* $T_n^{\mathsf{h}_n}$*. Let* E_q *be the event that* A^{h_n} *makes no more than* q *total queries.*

Claim 9 *If* $E_1 \cap E_2 \cap E_q$*, then*

$$\sum_{r=1}^{s_{n+1}} p_r \geq \ln 2 \cdot \left(\frac{1}{2} - \epsilon\right) \cdot \frac{1}{2} \cdot n^2 \, .$$

Proof. Since E_2 holds, every query $T_0, \ldots, T_n$ gets made, in the correct order. Since E_q holds, all these queries happen no later than query q, thus ensuring that Claim 8 applies and each t_k is finite. Moreover, by definition of p_r in Equation 1, $p_r \geq 1$ and $p_0 = 1$. Therefore, we can apply Claim 6 to the execution of A^{h_n} to get the desired result. □

CONVERTING FROM $\sum p_r$ TO $\mathsf{cc}_{\mathsf{mem}}$ Now we need to convert from $\sum p_r$ to $\sum m_r$.

Claim 10 *For every $r > 0$,*

$$3m_r \geq p_r \cdot (w - 2\log n - \log q - 1)\,.$$

Proof. By definition of p_r, we have that

$$p_r = \left\lceil \frac{m_r + 1 + \log q}{w - 2\log n - \log q - 1} + 1 \right\rceil \leq \frac{m_r + 1 + \log q}{w - 2\log n - \log q - 1} + 2,$$

since the ceiling adds at most 1. Therefore,

$$(p_r - 2) \cdot (w - 2\log n - \log q - 1) \leq m_r + 1 + \log q,$$

and thus

$$\begin{aligned} m_r &\geq (p_r - 2) \cdot (w - 2\log n - \log q - 1) - \log q - 1 \\ &= p_r \cdot (w - 2\log n - \log q - 1) - 2 \cdot (w - 2\log n - \log q - 1) - \log q - 1 \\ &= p_r \cdot (w - 2\log n - \log q - 1) - 2 \cdot (w - 2\log n - 0.5\log q - 0.5). \end{aligned}$$

Since $m_r \geq w \geq w - 2\log n - 0.5\log q - 0.5$, we can increase the left-hand side by $2 \cdot m_r$ and the right-hand side by $2 \cdot (w - 2\log n - 0.5\log q - 0.5)$ and the inequality still holds; therefore

$$3m_r \geq p_r \cdot (w - 2\log n - \log q - 1)\,.$$

□

.

Lemma 2. *Assuming $E_1 \cap E_2$ (see Definition 7), for any integer q, either A^{h_n} makes more than q queries (and thus $\mathsf{cc}_{\mathsf{mem}}(A^{\mathsf{h}_n}) > qw$ by definition) or*

$$\mathsf{cc}_{\mathsf{mem}}(A^{\mathsf{h}_n}(X)) \geq \frac{\ln 2}{6} \cdot \left(\frac{1}{2} - \epsilon\right) \cdot n^2 \cdot (w - 2\log n - \log q - 1)\;.$$

Proof. We observe that if A^{h_n} makes no more than q queries, then $E_1 \cap E_2 \cap E_q$ hold, and we can combine Claims 9 and 10 to get

$$\begin{aligned} \mathsf{cc}_{\mathsf{mem}}(A^{\mathsf{h}_n}(X)) = \sum_{r=1}^{s_{n+1}} m_r &\geq \frac{1}{3} \cdot \sum_{r=1}^{s_{n+1}} p_r \cdot (w - 2\log n - \log q - 1) \\ &\geq \frac{\ln 2}{3} \cdot \left(\frac{1}{2} - \epsilon\right) \cdot \frac{1}{2} \cdot n^2 \cdot (w - 2\log n - \log q - 1)\;. \end{aligned}$$

□

All that remains is to show a lower bound for the probability of $(E_1 \cap E_2 \cap E_q) \cup E_q$, and to argue that h_n is uniform, because the statement we are trying to prove is concerned with A^{h} for uniform h rather than with A^{h_n}.

The Uniformity of Oracles $\{\mathsf{h}_k\}$. Instead of proving h_n is uniform, we prove a stronger result that h_k is uniform for every k, which we will need later, anyway.

Claim 11 *For every k $(0 \le k \le n)$, h_k is uniform.*

Proof. We will prove this claim by induction. h_0 is uniform by definition. Assume h_j is uniform. Either $\mathsf{h}_j \in \mathsf{colliding}_j$ or not (see Definition 2). By Definition 3, if $\mathsf{h}_j \in \mathsf{colliding}_j$, then $\mathsf{h}_{j+1} = \mathsf{h}_j \in \mathsf{colliding}_j$; else, $\mathsf{h}_{j+1} \notin \mathsf{colliding}_j$. Thus, it remains to show that if h_j is uniform over the complement of $\mathsf{colliding}_j$, then so is h_{j+1}. Instead of thinking of h_j as being chosen all at once, think of it as being lazily sampled, one output at a time, starting with input X_0 and proceeding to inputs $X_1^{\mathsf{h}_j}, \ldots, X_{n-1}^{\mathsf{h}_j}, T_1^{\mathsf{h}_j}, \ldots, T_{j-1}^{\mathsf{h}_j}$, with outputs being selected uniformly from the set of outputs that do not cause h_j to fall into $\mathsf{colliding}_j$. Since $T_j^{\mathsf{h}_j}$ is distinct from all of these inputs (because $\mathsf{h}_j \notin \mathsf{colliding}_j$), the value of $\mathsf{h}_j(T_j^{\mathsf{h}_j})$ does not affect whether h_j is in $\mathsf{colliding}_j$, and thus the output $S_j^{\mathsf{h}_j}$ of h_j on $T_j^{\mathsf{h}_j}$ is uniform in $\{0, 1, \ldots, \lfloor 2^w/n \rfloor \cdot n - 1\}$ and independent of the rest of $T_j^{\mathsf{h}_j}$. To produce h_{j+1}, we replace $S_j^{\mathsf{h}_j}$ with $\mathrm{changeModn}(S_j^{\mathsf{h}_j}, c_{j+1})$ for uniformly random c_{j+1}, which does not change its distribution. □

Probability Analysis. Now we show the lower bound for the probability of $(E_1 \cap E_2 \cap E_q) \cup E_q$.

We already defined what it means for a challenge c_k to be hard (see Definition 5).

Definition 8. *Define every challenge c_k $(0 \le c_k < n)$ to be "bad" if $\mathsf{h}_{k-1} \in \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$ (see Definitions 2 and 4). We denote by E_3 the event that the number of challenges that are hard or bad is at least $n(\frac{1}{2} - \epsilon)$. Let E_4 be the event that for every k $(0 \le k < n)$, $\mathsf{h}_k \notin \mathsf{predictable}$ (see Definition 4).*

Note that $E_2 \cap E_4$ implies that no challenge is bad, and therefore $E_2 \cap E_3 \cap E_4 \Rightarrow E_1$: i.e., if there are more than $n(\frac{1}{2} - \epsilon)$ bad or hard challenges, and no challenge is bad, then there are more than $n(\frac{1}{2} - \epsilon)$ hard challenges. Thus, $E_2 \cap E_3 \cap E_4 \Rightarrow E_1 \cap E_2$. We will give a lower bound of $\Pr[E_2 \cap E_q \cap E_3 \cap E_4]$, and thus obtain a lower bound for $\Pr[E_1 \cap E_2 \cap E_q]$.

Since E_2, E_q, E_3, and E_4 depend on the oracles h_k, we need to understand their distribution; as we have proved before in Claim 11, oracle h_k is uniform for every k $(0 \le k \le n)$.

The upper bound on $\bar{E}_2 \cap E_q$. E_2 is the event that $\mathsf{h}_k \notin \mathsf{colliding}_k \cup \mathsf{wrongOrder}_k$ (see Definitions 2 and 6) for every k, and $T_n^{\mathsf{h}_n}$ is queried by A^{h_n}. We would like to worry about E_2 not for every oracle $\mathsf{h}_0 \ldots \mathsf{h}_n$, but only for h_n (to save a factor of n in the union bound). The following claim will help us do that.

Claim 12 *Given adversary A, if $h_k \in \mathsf{colliding}_k \cup \mathsf{wrongOrder}_k$ for some k ($0 \leq k < n$), then $h_{k+1} \in \mathsf{colliding}_{k+1} \cup \mathsf{wrongOrder}_{k+1}$, and therefore $h_n \in \mathsf{colliding}_n \cup \mathsf{wrongOrder}_n$.*

Proof. If $\mathsf{h}_k \in \mathsf{colliding}_k$, then $h_{k+1} = h_k$ and we are done. Else, let $T_j^{h_k}$ for $0 < j \leq k$ be a violation of the correct order: the query $T_i^{\mathsf{h}_k}$ does not occur before the first occurrence of $T_j^{\mathsf{h}_k}$ (for some $i < j$) when A^{h_k} is run; moreover, if there are multiple violations of the correct order, pick the one that occurs earliest. Note that $T_k^{\mathsf{h}_k}$ is first queried by A^{h_k} no earlier than $T_j^{\mathsf{h}_k}$ is: if $j = k$, this statement is a tautology, and if $j < k$, this statement is true because else the query of $T_k^{\mathsf{h}_k}$ would be an earlier violation. Since $\mathsf{h}_k \notin \mathsf{colliding}_k$ and $i, j \leq k$, we have $T_i^{\mathsf{h}_k} = T_i^{\mathsf{h}_{k+1}}$ and $T_j^{\mathsf{h}_k} = T_j^{\mathsf{h}_{k+1}}$. Moreover, since $T_k^{\mathsf{h}_k}$ is first queried by A^{h_k} no earlier than $T_j^{\mathsf{h}_k}$, and the computation of A^{h_k} and $A^{\mathsf{h}_{k+1}}$ proceeds identically until T_k is first queried, the same violation of the correct order occurs in the computation of $A^{\mathsf{h}_{k+1}}$: the value of $T_i^{\mathsf{h}_{k+1}}$ is not queried before $T_j^{\mathsf{h}_{k+1}}$ is. □

Thus, to upper bound $\Pr[\bar{E}_2]$, it is enough to bound the probability that a uniformly chosen random oracle (h_n) is in $\mathsf{colliding}_n \cup \mathsf{wrongOrder}_n$, or $A_n^{\mathsf{h}_n}$ does not query T_n^{h}. Recall from Definition 2 that the set $\mathsf{colliding}_n = \mathsf{colliding}_n^* \cup \mathsf{roundingProblem}_{n-1}$.

Claim 13 $|\mathsf{colliding}_n^*| \leq \mathcal{S} \cdot 1.5n^3 2^{-w}$.

Proof. To obtain an upper bound of $|\mathsf{colliding}_n^*|$, we consider three possible cases:

- The first colliding pair is $X_i^{\mathsf{h}} = X_j^{\mathsf{h}}$ such that $0 \leq i < j < n$. Fix (i, j), it implies that $X_{i-1}^{\mathsf{h}} \neq X_{j-1}^{\mathsf{h}}$ and
$$\mathsf{h}\left(X_{i-1}^{\mathsf{h}}\right) = \mathsf{h}\left(X_{j-1}^{\mathsf{h}}\right) ,$$
there are at most $\mathcal{S}2^{-w}$ such oracles; and there are at most $n^2/2$ possible colliding pairs (i, j).
- The first colliding pair is $X_i^{\mathsf{h}} = T_j^{\mathsf{h}}$, such that $0 \leq i < n$ and $0 < j \leq n$. This means $X_i = \mathsf{h}(T_{j-1}) \oplus X_{S_{j-1} \bmod n}$; thus $\mathsf{h}(T_{j-1})$ has to be in the set $\{(X_i \oplus X_\ell)\}_{0 \leq \ell < n}$. Since T_{j-1} is distinct from X's (because (i, j) is the first colliding pair), by fixing X's values first and counting how many values of $\mathsf{h}(T_{j-1})$ are in $\{(X_i \oplus X_\ell)\}_{0 \leq \ell < n}$, we know there are at most $\mathcal{S}n2^{-w}$ oracles h satisfying the above. And there are at most n^2 possible colliding pairs (i, j).
- The first colliding pair is $T_i^{\mathsf{h}} = T_j^{\mathsf{h}}$ such that $0 < i < j \leq n$. This means
$$S_{i-1} \oplus X_{S_{i-1} \bmod n} = S_{j-1} \oplus X_{S_{j-1} \bmod n} ,$$
i.e.,
$$\mathsf{h}(T_{i-1}^{\mathsf{h}}) \oplus X_{S_{i-1} \bmod n} = \mathsf{h}(T_{j-1}^{\mathsf{h}}) \oplus X_{S_{j-1} \bmod n} ,$$
since T_{j-1}^{h} is distinct to all X's and T_{i-1}^{h} (because (i, j) is the first colliding pair), and $\mathsf{h}(T_{j-1}^{\mathsf{h}})$ should be in the set $\{(\mathsf{h}(T_{i-1}^{\mathsf{h}}) \oplus X_{S_{i-1} \bmod n} \oplus X_\ell)\}_{0 \leq \ell < n}$ in

order to satisfy the equality. By fixing X's values and $\mathsf{h}(T^{\mathsf{h}}_{i-1})$ first and counting how many values of $\mathsf{h}(T^{\mathsf{h}}_{j-1})$ are in $\{(\mathsf{h}(T^{\mathsf{h}}_{i-1}) \oplus X_{S_{i-1} \bmod n} \oplus X_\ell)\}_{0 \le \ell < n}$, we know there are at most $\mathcal{S}n2^{-w}$ oracles h satisfying above. And there can be at most $n(n-1)/2$ colliding pairs (i, j).

From above, $|\mathsf{colliding}^*_n| \le \mathcal{S}(.5n^2 + n^3 + .5n^2(n-1))2^{-w} = \mathcal{S} \cdot 1.5n^3 2^{-w}$. □

Second, we bound the number of oracles $\mathsf{h} \in \mathsf{roundingProblem}_{n-1} \setminus \mathsf{colliding}^*_n$ (Definition 1), which is enough (together with the previous bound on the size of $\mathsf{colliding}^*_n$), to bound the size of $\mathsf{colliding}_n$.

Claim 14 *$|\mathsf{roundingProblem}_{n-1} \setminus \mathsf{colliding}^*_n| \le \mathcal{S}n(n-1)2^{-w}$.*

Proof. For each i, $0 \le i \le n-1$, we will bound the size of $\mathsf{roundingProblem}_i \setminus \mathsf{roundingProblem}_{i-1} \setminus \mathsf{colliding}^*_n$. The intuitive idea is that $\mathsf{h}(T_i)$ can take only $n-1$ out of equiprobable 2^w values in order for h to get into this set. However, to make this idea precise, we need to first fix T_i that does not collide with anything (otherwise, it is not true that all values are equiprobable). To do so, fix any sequence of w-bit strings $X_1, \ldots, X_{n-1}, S_0, \ldots S_{i-1}$ and let $T_i = S_{i-1} \oplus X_{S^{\mathsf{h}}_{i-1} \bmod n}$. Partition the set of all oracles into subsets $H(X_1, \ldots, X_{n-1}, S_0, \ldots S_{i-1}) = \{\mathsf{h} \text{ s. t. } X^{\mathsf{h}}_1 = X_1, \ldots, X^{\mathsf{h}}_{n-1} = X_{n-1}, S^{\mathsf{h}}_0 = S_0, \ldots S^{\mathsf{h}}_{i-1} = S_{i-1}\}$. If T_i is equal to one of $X_0, \ldots,$ $X_{n-1}, T_1, \ldots, T_{i-1}$, then every element of this subset is in $\mathsf{colliding}^*_n$. Else, this subset can further partitioned into 2^w equally sized subsets depending on $\mathsf{h}(T_i)$, and only elements of (at most $n-1$) subsets for which $\mathsf{h}(T_i) > \lfloor 2^w/n \rfloor \cdot n - 1$ can be in $\mathsf{roundingProblem}_i \setminus \mathsf{roundingProblem}_{i-1}$. Taking the union bound over all i, we get the claimed result. □

Claim 15 *Given adversary A, the number of oracles $\mathsf{h} \notin \mathsf{colliding}^*_n$ such that E_q (see Definition 7) holds and A^{h_n} does not query T^{h}_n is no more than $\mathcal{S}(\Pr[E_q] - \chi_q + 2^{-w})$, where χ_q is the probability (for a uniform h) that A^{h} is successful and makes no more than q queries.*

Proof. First, we will bound the number of oracles outside of $\mathsf{colliding}^*_n$ for which A successfully outputs S_n without querying T_n. We use a similar approach to the proof of Claim 14 to make sure T_n is well-defined and can take any of 2^w equally likely values. Define the subset $H(X_1, \ldots, X_{n-1}, S_0, \ldots S_{n-1})$ the same way as in Claim 14. If T_n collides with one of $X_1, \ldots, X_{n-1}, T_1, \ldots T_{n-1}$, then every element in the subset is $\mathsf{colliding}^*_n$. Else, partition this subset further into subsets H_α according to the answers α given to queries of A. What A does, including whether A queries T_n, depends only α, and if A does not query T_n, then H_α can be partitioned into 2^w equal-size parts according to the value of $\mathsf{h}(T_n)$. Since the output of A is determined by α, A can be successful for only one of those parts. Thus, there are at most $\mathcal{S} \cdot 2^{-w}$ oracles outside of $\mathsf{colliding}^*_n$ for which A is successful but does not query T_n.

In addition, there are $\mathcal{S} \cdot (\Pr[E_q] - \chi_q)$ oracles for which A^{h} makes no more than q queries but fails.

□

Finally, we will bound the number of oracles h such that $\mathsf{h} \in \mathsf{wrongOrder}_n \setminus \mathsf{colliding}^*_n$ (Definitions 6, 2) and E_q (Definition 7).

Claim 16 *Given adversary A, the number of oracles h such that E_q holds and $\mathsf{h} \in \mathsf{wrongOrder}_n \setminus \mathsf{colliding}^*_n$ is no more than $\mathcal{S}qn^2 2^{-w}$.*

Proof. Let j, $0 < j \leq n$ be the smallest value for which there exists some $0 \leq i < j$ such that T^{h}_i has not been queried by A^{h} by the time T^{h}_j is. Note that then T^{h}_{j-1} has also not been queried by the time of the T^{h}_j query (because either $j = 1$ or else j would not be the smallest, since $j-1$ also satisfies the condition). In order for any given query t to be equal to T^{h}_j, the value $S^{\mathsf{h}}_{j-1} = \mathsf{h}(T^{\mathsf{h}}_{j-1})$ needs to at least satisfy $S^{\mathsf{h}}_{j-1} = t \oplus X_c$ for some c. However, t is independent of the value S^{h}_{j-1} as long as query T^{h}_{j-1} hasn't been by the time t is made. Thus, for every one of q possible queries, there are at most n values S^{h}_{j-1} (out of 2^w possible ones) that will make this query equal to T^{h}_j. (To formalize this argument, we need to make sure T^{h}_j is well-defined and can take 2^w possible values, which we do in exactly the same way as in Claim 15, using the fact that $\mathsf{h} \notin \mathsf{colliding}^*_n$. We omit this formalization to avoid repetition.)

Taking the union bound over all q queries and n possible values of j gives the desired result. □

From all above, $\Pr[\bar{E}_2 \cap E_q]$ is upper bounded by

$$\Pr[E_q] - \chi_q + (1.5n^3 + n(n-1) + 1 + qn^2) \cdot 2^{-w} \leq \Pr[E_q] - \chi_q + ((q+1)n^2 + 1.5n^3) \cdot 2^{-w} \,,$$

where χ_q is the probability (for a uniform h) that A^{h} is successful and makes no more than q queries.

The upper bound of $\bar{E}_3$. Recall that according to Definition 8, E_3 is the event that the number of challenges that are hard or bad is at least $n(\frac{1}{2} - \epsilon)$.

Claim 17 $\Pr[\bar{E}_3] \leq e^{-2\epsilon^2 n}$.

Proof. We bound the probability of $\bar{E}_3$ by a slightly more complicated version of the Hoeffding bound used in the proof of Theorem 4. Recall that a challenge c_k $(0 \leq c_k < n)$ is "bad" if $\mathsf{h}_{k-1} \in \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$. Define random variable $F_k = -1$ if challenge c_k is hard or bad, and $F_k = 1$ otherwise; We know that for *any* (particular fixing of the values of) $\mathsf{h}_0, c_1, \ldots, c_{k-1}$, which defines a particular fixing $(F_1, \ldots, F_{k-1}) = (f_1, \ldots, f_{k-1})$,

$$\Pr[F_k = -1 | (F_1, \ldots, F_{k-1}) = (f_1, \ldots, f_{k-1})] \geq \frac{1}{2} \,, \tag{2}$$

because if $\mathsf{h}_{k-1} \notin \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$, then Lemma 1 applies, and otherwise c_k is always bad.

Define the random variable $Y_k = F_1 + \cdots + F_k$ for $0 < k \leq n$; it is easy to see that $\mathbb{E}[Y_k]$ is finitely upper bounded for every $0 < k \leq n$. For every $(Y_1, \ldots Y_i) = (y_1, \ldots y_i)$ we have that

$$\mathbb{E}[Y_{i+1} | (Y_1, \ldots, Y_i) = (y_1, \ldots, y_i)] = \mathbb{E}[y_i + F_{i+1} | (Y_1, \ldots, Y_i) = (y_1, \ldots, y_i)] \leq y_i \,,$$

where the last inequality holds because of Equation 2. Therefore, we have that

$$\mathbb{E}[Y_{i+1}|Y_1,\dots,Y_i] = \mathbb{E}[Y_i + F_{i+1}|Y_1,\dots,Y_i] \le Y_i \ ,$$

and thus $\{Y_k\}$ is a super-martingale. Set $Y_0 = 0$, note $|Y_k - Y_{k-1}| \le 1$ for every $0 < k \le n$, by Azuma-Hoeffding's inequality [Azu67] we get that

$$\Pr[Y_n - Y_0 \ge 2\epsilon n] \le e^{-2\epsilon^2 n} \ .$$

Since $Y_0 = 0$ and Y_n is equal to the difference between the number of challenges that are not hard or bad and the number of challenges that are hard or bad, we obtain that with probability at least $1 - e^{-2\epsilon^2 n}$, the number of challenges that are hard or bad (over n challenges) is at least $n(\frac{1}{2} - \epsilon)$. Thus $\Pr[\bar{E_3}] \le e^{-2\epsilon^2 n}$. □

The upper bound of $\bar{E_4}$. Recall from Definition 8 that E_4 is the event that for every $0 \le k < n$, $\mathsf{h}_k \notin \mathsf{predictable}$. Recall that $\mathsf{predictable}$ (Definition 4) is the set of random oracles for which a predictor $\mathcal{P}$ successfully predicts the output of h on p_r distinct inputs given the M-bit input state and an additional $(p_r(2\log n + \log q + 1) + \log q)$-bit hint. Using the same technique explained in the proof of Claim 3, we can get a similar bound. Note now that here

$$p_r = \lceil (m_r + 1 + \log q)/(w - 2\log n - \log q - 1) + 1 \rceil \ ,$$

and following the same lines of the proof and the computation in Claim 3, we get a new bound that for each h_k,

$$\Pr[\mathsf{h}_k \in \mathsf{predictable}] \le 2(n-1)n^2 q \cdot 2^{-w} \ .$$

Indeed, to see why, note that when p is fixed, the number of possible input states and hints is at most $2^{(M_p+1)+\log q + p(2\log n + \log q + 1)}$; this gives us at most

$$\mathcal{S} \cdot 2^{(M_p+1+\log q)+p(2\log n+\log q+1-w)} \le \mathcal{S} \cdot 2^{2\log n + \log q + 1 - w}$$

random oracles in $\mathsf{predictable}$ since $(M_p+1+\log q) \le (p-1)(w-2\log n - \log q - 1)$. Therefore, by adding up bounds for all possible p from 1 to $n-1$, $|\mathsf{predictable}|$ does not exceed $(n-1)\mathcal{S}2^{2\log n+\log q+1-w} = \mathcal{S}2(n-1)n^2 q 2^{-w}$. Finally, by the union bound over the n oracles $\{\mathsf{h}_k\}$ (each uniformly distributed), we get $\Pr[\bar{E_4}] \le 2q(n-1)n^3 \cdot 2^{-w}$.

Final wrap-up. Recall that χ is the probability that A^{h} succeeds and χ_q is the probability that A^{h} succeeds and makes at most q queries. From all above, we have

$$\begin{aligned}\Pr[E_2 \cap E_q \cap E_3 \cap E_4] &\ge \Pr[E_q] - \Pr[\bar{E_2} \cap E_q] - \Pr[\bar{E_3}] - \Pr[\bar{E_4}] \\ &\ge \chi_q - ((q+1)n^2 + 1.5n^3 + 2q(n-1)n^3) \cdot 2^{-w} - e^{-2\epsilon^2 n} \\ &\ge \chi_q - 2qn^4 \cdot 2^{-w} + (2q - \frac{q}{n} - \frac{1}{n} - 1.5)n^3 \cdot 2^{-w} - e^{-2\epsilon^2 n} \\ &\ge \chi_q - 2qn^4 \cdot 2^{-w} - e^{-2\epsilon^2 n} \ ,\end{aligned}$$

because if $n \ge 2$ and $q \ge 2$, then $1/n + 1.5 \le 2 \le q$.

Note that $\chi_q + \Pr[E_q] \ge \chi$. Therefore, $\Pr[(E_2 \cap E_q \cap E_3 \cap E_4) \cup E_q] \ge \chi - 2qn^4 \cdot 2^{-w} - e^{-2\epsilon^2 n}$. Combining this statement with the result of Lemma 2 and the discussion following Definition 8, we get the result of Theorem 5.

Acknowledgments

We thank Chethan Kamath and Jeremiah Blocki for helpful discussions and Jeremiah Blocki for his careful reading of our proof.

Leonid Reyzin gratefully acknowledges the hospitality and support of IST Austria, where most of this work was performed, and the hospitality of École normale supérieure, Paris.

The work was partially supported by the following U.S. NSF grants: Binyi Chen by CNS-1423566, CNS-1528178, and CNS-1514526; Stefano Tessaro by CNS-1423566, CNS-1528178, CNS-1553758 (CAREER), and IIS-1528041; Leonid Reyzin by 1012910, 1012798, and 1422965. Moreover, Joël Alwen and Krzysztof Pietrzak were supported by the European Research Council consolidator grant 682815-TOCNeT.

References

AB16. Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In *Advances in Cryptology CRYPTO'16*. Springer, 2016.

ABMW05. Martín Abadi, Michael Burrows, Mark S. Manasse, and Ted Wobber. Moderately hard, memory-bound functions. *ACM Trans. Internet Techn.*, 5(2):299–327, 2005.

ABW03. Martín Abadi, Michael Burrows, and Ted Wobber. Moderately hard and memory-bound functions. In *NDSS 2003*. The Internet Society, February 2003.

ACK$^+$16a. Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. On the complexity of Scrypt and proofs of space in the parallel random oracle model. Cryptology ePrint Archive, Report 2016/100, 2016. http://eprint.iacr.org/2016/100.

ACK$^+$16b. Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. On the complexity of scrypt and proofs of space in the parallel random oracle model. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 358–387. Springer, Heidelberg, May 2016.

AS15. Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 595–603. ACM Press, June 2015.

Azu67. Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Math. J. (2)*, 19(3):357–367, 1967.

BDK16. Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2 password hash. Version 1.3, 2016. https://www.cryptolux.org/images/0/0d/Argon2.pdf.

Cha11. Charles Lee. Litecoin, 2011.

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

DGN03. Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 426–444. Springer, Heidelberg, August 2003.

DN93. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, Heidelberg, August 1993.

DNW05. Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 37–54. Springer, Heidelberg, August 2005.

FM97. Matthew K. Franklin and Dahlia Malkhi. Auditable metering with lightweight security. In Rafael Hirschfeld, editor, *FC'97*, volume 1318 of *LNCS*, pages 151–160. Springer, Heidelberg, February 1997.

Hoe63. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

JB99. Ari Juels and John G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS'99*. The Internet Society, February 1999.

JJ99. Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security*, CMS '99, pages 258–272, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V.

Per09. C. Percival. Stronger key derivation via sequential memory-hard functions. In *BSDCan 2009*, 2009.

PHC. Password hashing competition. https://password-hashing.net/.

PJ16. C. Percival and S. Josefsson. The scrypt Password-Based Key Derivation Function. RFC 7914 (Informational), August 2016.

RSW96. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.

vABHL03. Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 294–311. Springer, Heidelberg, May 2003.

A On Percival's Proof

We note that Percival [Per09] claims a weaker result than the one in our main theorem, similar in spirit to our single-shot trade-off theorem (Theorem 2 above), in that it considers only a single random challenge, as well as an overall upper bound on the size of the initial state. Also, the proof technique of [Per09] may, at first, somewhat resemble the one used in Theorem 2, where multiple copies of the adversary are run on all possible challenges. In contrast to both this work and that of [ACK$^+$16b], however, Percival considers adversaries with only a limited amount a parallelism.

Upon closer inspection, however, we have found serious problems with the proof in [Per09]. In more detail, the proof considers an adversary running in two stages. In the preprocessing stage the adversary gets input B and access to h and must eventually output an arbitrary state (bit-string) σ. In the second phase n copies of the adversary are run in parallel. For $x \in [0, n-1]$ the x^{th} copy is given challenge x, state σ and access to h. Its goal is to produce output $\mathsf{h}^x(B)$. The main issue with the proof stems from the fact that information about h contained within σ is never explicitly handled. Let us be a bit more concrete.

The proof looks in particular at the set $\overline{R}_i$ of all $i \in [n]$ of all values U for which some copy of the adversary queries $\mathsf{h}(U)$ within the first i steps. Here, some key aspects remain undefined. For instance, it is unclear whether the initial time step in the second phase is 0 or 1, and consequently, there is also no clear definition of the contents of the set $\overline{R}_0$. We briefly discuss now why, no matter how we interpret $\overline{R}_0$, the technique does not imply the desired statement.

Suppose we assume that $\overline{R}_0$ is the set of queries to h made by the adversary in this first step of the second stage. In particular, for all i, the set $\overline{R}_i$ contains only queries to h made during second phase of the execution. However this creates a serious problem. At a (crucial) later step in the proof it is claimed that if $\mathsf{h}^{x-1}(B) \notin \overline{R}_{i-1}$, then the probability that $\mathsf{h}^x(B)$ is queried at the i-th step is the same as simply guessing $\mathsf{h}^x(B)$ out of the blue (a highly unlikely event). But this statement is now incorrect as it ignores potential information contained in the state σ. For example σ may even contain $\mathsf{h}^x(B)$ explicitly making it trivial to query h at that point at any time i regardless of the contents of $\overline{R}_{i-1}$.

Suppose instead that we assume the time of the second phase begins at 1 leaving $\overline{R}_0$ open to interpretation. Setting $\overline{R}_0 = \emptyset$ leads to the exact same problem as before. So instead, in an attempt to avoid this pitfall, we could let $\overline{R}_0$ be the set of queries made during the pre-computation stage. Indeed, if $\mathsf{h}^{x-1}(B) \notin \overline{R}_i$ then that means $\mathsf{h}^{x-1}(B)$ was not queried while σ was being prepared and so (whp) σ contains no information about $\mathsf{h}^x(B)$ avoiding the previous problem. Yet here too we run in to issues. Consider the following adversary $\mathcal{A}$: In the preprocessing stage $\mathcal{A}$ makes all queries $\mathsf{h}^x(B)$ for $x \in [0, n-1]$ and then generates some state σ (what this state really is, and how the adversary proceeds in the second stage is somewhat irrelevant). In particular for this adversary, for all i the set $\overline{R}_i$ already contains all relevant queries $\{\mathsf{h}^x(B) : x \in [0, n-1]\}$. Most of the remainder of the proof is concerned with upper bounding the expected size of $\overline{R}_i$. But in the case of $\mathcal{A}$ for each i we now have $\left|\overline{R}_i\right| \geq n$ which contradicts the bounds shown in the proof. Worse, when plugging in this new upper bound into the remaining calculations in the proof we would get that the expected runtime of each instance of $\mathcal{A}$ in the second phase is at least 0; an uninteresting result. Thus this too can not be the right interpretation. Unfortunately, we were unable to come up with any reasonable interpretation which results in an interesting statement being proven.

In conclusion, we note that the proof can be adapted to the randomized pebbling setting, as considered in [ACK$^+$16b]. However, we note that for this setting, [ACK$^+$16b] already contains a much simpler proof of such a single-shot trade-off theorem. We also note that Theorem 2 confirms that Percival's statement is in fact true, although using a very different proof technique.

B An Attack to the Repeated Challenges Pebbling Game.

We discuss more in detail the general bound of Theorem 4. Note first that the term

$$\ln\left(1+\left(\frac{\mathsf{p}_{\min}}{\mathsf{p}_0}\right)^{\frac{1}{Q\left(\frac{1}{2}-\epsilon\right)}}\right)$$

can be small when both the ratio $\frac{\mathsf{p}_{\min}}{\mathsf{p}_0}$ and the number of challenges Q are small. For example, when $\frac{\mathsf{p}_{\min}}{\mathsf{p}_0} = 1/n$ and $Q = \log n/\log\log n$, the above term is less than $1/\log n$. (Notice that when the number of challenges $Q = n$, the above term has a constant lower bound around $1/2$, and the lower bound of $\mathsf{cc}_{\mathsf{peb}}$ is still $\Omega(n^2)$.) We want to discuss here why this phenomenon is inherent, by giving a concrete pebbling strategy when $\mathsf{p}_0 > \mathsf{p}_{\min}$.[8] The cumulative pebbling complexity of the attack is no more than

$$\mathsf{p}_0 + nQ\cdot\left(\frac{\mathsf{p}_{\min}}{\mathsf{p}_0}\right)^{1/Q} + \frac{2n}{\mathsf{p}_{\min}}\cdot\frac{1}{\left(\frac{\mathsf{p}_0}{\mathsf{p}_{\min}}\right)^{1/Q}-1}\ .$$

Notice that since $\ln(1+x)\approx x$ when x is small, when the term $\left(\frac{\mathsf{p}_{\min}}{\mathsf{p}_0}\right)^{1/Q}$ is small, the cumulative complexity of the attack is not far from the general lower bound we have in Theorem 4.

Roughly speaking, the strategy is as follows: the adversary can prepare a lot of pebbles in the beginning, and then scale down the number of pebbles quickly after she knows the next challenge; by the end of the game, the number of pebbles is reduced down to $\mathsf{p}_{\min}$.

More precisely, set the scale parameter $c = \left(\frac{\mathsf{p}_0}{\mathsf{p}_{\min}}\right)^{1/Q}$. The attack works as follows: initially, the adversary puts p_0 equidistant pebbles on nodes

$$\left(0, \left\lceil\frac{n}{\mathsf{p}_0}\right\rceil, \ldots, (\mathsf{p}_0-1)\cdot\left\lceil\frac{n}{\mathsf{p}_0}\right\rceil\right)\ .$$

Notice that every node $0\le i<n$ has a nearest pebbled predecessor whose distance to i is no more than n/p_0. After knowing the first challenge, the adversary keeps at most $\left\lceil\frac{\mathsf{p}_0}{c}\right\rceil+1$ pebbles and immediately removes all other pebbles. In particular, among the current p_0 pebbles, the adversary keeps pebble x_1^* which is the nearest pebbled predecessor to the first challenge node; additionally, for every node $s_i = i\cdot\left\lceil\frac{cn}{\mathsf{p}_0}\right\rceil$ where $0\le i<\left\lceil\frac{\mathsf{p}_0}{c}\right\rceil$, the adversary keeps the pebble

[8] Note when $\mathsf{p}_0 = \mathsf{p}_{\min}$, we can simply keep pebble at node 0 and move the pebble to the challenge node in less than n steps after the challenge is given. Since the number of pebbles in every moment is no more than 2, the cumulative pebbling complexity is $\mathsf{p}_0 + 2nQ$.

which is the nearest pebbled predecessor to s_i.[9] Then in each step, for every pebble at node i such that

- i is not the first challenge node.
- For any $0 \leq j < \lceil \frac{\mathsf{p}_0}{c} \rceil$, $i \neq s_j$.

the adversary generates a pebble at node $i+1$, and removes pebble i (in parallel).[10] Intuitively, after each step, every pebble is becoming one step closer to their target node. The process ends whenever the first challenge node is pebbled, and every node is pebbled in set

$$\left(0, \left\lceil \frac{cn}{\mathsf{p}_0} \right\rceil, \ldots, \left(\left\lceil \frac{\mathsf{p}_0}{c} \right\rceil - 1 \right) \cdot \left\lceil \frac{cn}{\mathsf{p}_0} \right\rceil \right) .$$

Since every node $0 \leq i < n$ has a nearest pebbled predecessor whose distance is no more than n/p_0, it is easy to see that the process takes at most n/p_0 steps[11].

More generally, just before the k^{th} challenge is given, pebbles are put on nodes

$$\left(0, \left\lceil \frac{c^{k-1}n}{\mathsf{p}_0} \right\rceil, \ldots, \left(\left\lceil \frac{\mathsf{p}_0}{c^{k-1}} \right\rceil - 1 \right) \cdot \left\lceil \frac{c^{k-1}n}{\mathsf{p}_0} \right\rceil \right) .$$

Notice that every node $0 \leq i < n$ has a nearest pebbled predecessor whose distance to i is no more than $\frac{c^{k-1}n}{\mathsf{p}_0}$. After knowing the k^{th} challenge, the adversary keeps at most $\lceil \frac{\mathsf{p}_0}{c^k} \rceil + 1$ pebbles and immediately removes all other pebbles. In particular, among the current $\lceil \frac{\mathsf{p}_0}{c^{k-1}} \rceil$ pebbles, the adversary keeps pebble x_k^* which is the nearest pebbled predecessor to the k^{th} challenge node; additionally, for every node $s_i = i \cdot \left\lceil \frac{c^k n}{\mathsf{p}_0} \right\rceil$ where $0 \leq i < \lceil \frac{\mathsf{p}_0}{c^k} \rceil$, the adversary keeps the pebble which is the nearest pebbled predecessor to s_i. Therefore, with similar argument above, the adversary can use no more than $\frac{c^{k-1}n}{\mathsf{p}_0}$ steps to pebble the k^{th} challenge (thus can answer it and receive the next challenge in the next step), and put pebbles on nodes

$$\left(0, \left\lceil \frac{c^k n}{\mathsf{p}_0} \right\rceil, \ldots, \left(\left\lceil \frac{\mathsf{p}_0}{c^k} \right\rceil - 1 \right) \cdot \left\lceil \frac{c^k n}{\mathsf{p}_0} \right\rceil \right) .$$

During the period for pebbling the k^{th} challenge node, in every moment, the adversary keeps at most $\lceil \frac{\mathsf{p}_0}{c^k} \rceil + 1$ pebbles, and it takes the adversary at most

[9] Notice that for different $s_i < s_j$, the nearest pebbled predecessors to nodes s_i and s_j are different, because $s_j - s_i \geq n/\mathsf{p}_0$ and thus the interval $[s_i, s_j]$ has at least one pebble.

[10] There is a special case in which the nearest pebbled predecessor of the first challenge node and some node s_j is identical; in this case, the adversary still generates pebble at node $i+1$ even if $i = s_j$ or i is the node of the first challenge node.

[11] Note that we remove the redundant pebbles and move the remaining pebbles one step closer to target nodes simultaneously; instead of using one step for removing, and then one step for moving the remaining pebbles.

$\frac{c^{k-1}n}{\mathsf{p}_0}$ steps to pebble the challenge node. Thus we obtain an upper bound of the cumulative complexity during this period

$$\left(\left\lceil\frac{\mathsf{p}_0}{c^k}\right\rceil+1\right)\cdot\frac{c^{k-1}n}{\mathsf{p}_0}\ .$$

Therefore, after Q challenges, the upper bound of the adversary's cumulative pebbling complexity is

$$\begin{aligned}\mathsf{cc}_{\mathsf{peb}} &\leq \mathsf{p}_0+\sum_{k=1}^{Q}\left(\left\lceil\frac{\mathsf{p}_0}{c^k}\right\rceil+1\right)\cdot\frac{c^{k-1}n}{\mathsf{p}_0}\\ &\leq \mathsf{p}_0+\sum_{k=1}^{Q}\left(\frac{\mathsf{p}_0}{c^k}+2\right)\cdot\frac{c^{k-1}n}{\mathsf{p}_0}\\ &= \mathsf{p}_0+\frac{nQ}{c}+\frac{2n}{\mathsf{p}_0}\cdot\sum_{k=1}^{Q}c^{k-1}\leq \mathsf{p}_0+\frac{nQ}{c}+\frac{2n}{\mathsf{p}_0}\cdot\frac{c^Q}{c-1}\ ,\end{aligned}$$

plugging in $c=\left(\frac{\mathsf{p}_0}{\mathsf{p}_{\min}}\right)^{1/Q}>1$, we have

$$\mathsf{cc}_{\mathsf{peb}}\leq \mathsf{p}_0+nQ\cdot\left(\frac{\mathsf{p}_{\min}}{\mathsf{p}_0}\right)^{1/Q}+\frac{2n}{\mathsf{p}_{\min}}\cdot\frac{1}{\left(\frac{\mathsf{p}_0}{\mathsf{p}_{\min}}\right)^{1/Q}-1}\ .$$