Bandwidth-Hard Functions: Reductions and Lower Bounds

Jeremiah Blocki
Department of Computer Science, Purdue University
`jblocki@purdue.edu`

Peiyuan Liu
Department of Computer Science, Purdue University
`liu2039@purdue.edu`

Ling Ren
Department of Computer Science, University of Illinois at Urbana-Champaign
`renling@illinois.edu`

Samson Zhou
Department of Computer Science & Engineering, Texas A&M University
`samsonzhou@gmail.com`

January 22, 2024

Abstract

Memory Hard Functions (MHFs) have been proposed as an answer to the growing inequality between the computational speed of general purpose CPUs and ASICs. MHFs have seen widespread applications including password hashing, key stretching and proofs of work. Several metrics have been proposed to quantify the memory hardness of a function. Cumulative memory complexity (CMC) quantifies the cost to acquire/build the hardware to evaluate the function repeatedly at a given rate. By contrast, bandwidth hardness quantifies the energy costs of evaluating this function. Ideally, a good MHF would be both bandwidth hard and have high CMC. While the CMC of leading MHF candidates is well understood, little is known about the bandwidth hardness of many prominent MHF candidates.

Our contributions are as follows: First, we provide the first reduction proving that, in the parallel random oracle model (pROM), the bandwidth hardness of a data-independent MHF (iMHF) is described by the red-blue pebbling cost of the directed acyclic graph associated with that iMHF. Second, we show that the goals of designing an MHF with high CMC/bandwidth hardness are well aligned. Any function (data-independent or not) with high CMC also has relatively high bandwidth costs. Third, we prove that in the pROM the prominent iMHF candidates such as Argon2i, aATSample and DRSample are maximally bandwidth hard. Fourth, we prove the first unconditional tight lower bound on the bandwidth hardness of a prominent data-dependent MHF called Scrypt in the pROM. Finally, we show the problem of finding the minimum cost red-blue pebbling of a directed acyclic graph is NP-hard.

Keywords: memory-hard functions, energy cost, bandwidth hardness, graph pebbling, cumulative memory complexity, parallel random oracle model

1 Introduction

Memory Hard Functions (MHFs) [28, 1] are a crucial building block in the design of password hashing functions, moderately hard key-derivation functions and egalitarian proofs of work [19, 9]. For example, in password hashing we would like to ensure that it is prohibitively expensive for an offline attacker to evaluate the function millions or billions of times to check each password in a large cracking dictionary.

The development of improved Application Specific Integrated Circuits (ASICs) or Field Programmable Gate Arrays (FPGAs) for computing cryptographic hash functions such as SHA256 makes this goal increasingly challenging. For example, the Antminer S9, an ASIC Bitcoin [27] miner, is able to compute SHA256 hashes at a rate of 13.6 trillion hashes per second with energy consumption of only 1274 Joules per second (Watts). By contrast, the energy needed to compute SHA256 13.6 trillion times on a standard CPU would be about six orders of magnitude higher! In fact, Blocki *et al.* [15] recently argued that non-memory-hard key derivation functions (e.g., PBKDF2-SHA256 and BCRYPT) *cannot* provide sufficient protection against a rational (economically motivated) offline attacker without introducing unacceptably long authentication delays.

MHFs are based on the observation that memory costs (e.g., latency, bandwidth, energy consumption) tend to be equitable across different architectures. Thus, to develop an "egalitarian" function we want to design a function where evaluation costs are dominated by memory costs. Two of the most prominent approaches to measure the "evaluation cost" of MHFs are memory hardness [28, 8] and bandwidth hardness [30]. Memory hardness [28] seeks to quantify construction costs i.e., the cost to build/obtain the hardware necessary to compute the MHF. By contrast, bandwidth hardness [30] seeks to quantify the energy costs per evaluation i.e., the cost of running the hardware. Ideally, one would hope to design an MHF that is *both* bandwidth hard and memory hard.

Broadly speaking there are two types of MHFs: data-dependent memory hard functions (dMHFs) and data-independent memory hard functions (iMHFs). As the name suggests an iMHF induces a memory access pattern that is *independent* of the sensitive input (e.g., password), which makes them naturally resistant to certain side channel attacks e.g., cache-timing [10]. Meanwhile, while dMHFs with high memory/bandwidth hardness are potentially easier to construct [2, 6], they are also more vulnerable to side channel attacks. Argon2 [12], winner of the recently completed Password Hashing Competition [29], includes a data-independent mode of operation (Argon2i), a data-dependent mode (Argon2d) and a hybrid mode (Argon2id).

To a large extent, most of the recent cryptanalysis of MHF candidates has focused on memory hardness. In particular, *cumulative memory complexity* (CMC) [8] and the closely related metric *amortized area-time complexity* (aAT) [2, 4] aim to approximate the cost of constructing enough chips to evaluate the function T times per year. For example, if evaluating the function one time requires us to lock up 1GB of DRAM for 1 second then, *at minimum*, an attacker would need to buy roughly 32 (1GB) DRAM chips to evaluate the function a billion times per year. Alwen *et al.* [6] showed that the dMHF `scrypt` [28] has aAT complexity that scales quadratically with the running time parameter n i.e., the function has CMC $\Omega(n^2)$.[1] By contrast, Alwen and Blocki [2, 3] showed that *any* iMHF has cumulative memory complexity *at most* $\mathcal{O}\left(\frac{n^2 \log\log n}{\log n}\right)$ and they exhibited even stronger amortization attacks against Password Hashing Competition [29] (PHC) winner Argon2i [12] along with other candidate MHFs such as balloon hashing [17]. Blocki and Zhou [16] showed that Argon2i has CMC at most $\mathcal{O}\left(n^{1.767}\right)$ and at least $\tilde{\Omega}\left(n^{1.75}\right)$. Alwen *et al.* [5] also gave a theoretical construction of an iMHF with CMC *at least* $\Omega\left(\frac{n^2}{\log n}\right)$, which is essentially optimal in an asymptotic sense. More recently, Alwen *et al.* [4] designed two *practical* iMHFs called $\mathsf{DRSample}$ and $\mathsf{aATSample}$ with the same asymptotic complexity.

By contrast, the notion of *bandwidth-hardness* was only introduced recently [30] with the intention of lower bounding the energy required to evaluate the function[2]. Ren and Devadas [30] observed that metrics such as CMC or aAT do not provide an accurate picture of energy consumption. For example, certain types of memory consume very little energy when idle, but cache misses are costly because we must *retrieve* data from RAM. Memory Bound Functions [1] are functions whose computation *always requires* a large number of cache-misses regardless of computation time. Bandwidth hardness [30] relaxes this notion by requiring that any attacker who evaluates the function must either 1) incur a large number of expensive cache misses, or 2) must perform a larger (e.g., super-linear) amount of computation.

Ren and Devadas proposed to cryptanalyze an iMHF using a variant of the red-blue pebbling game in which red-moves (representing computation performed using data in cache) have a smaller cost c_r than

[1]This is the best possible lower bound for CMC. In particular, any MHF that can be evaluated in time $\mathcal{O}(n)$ on a sequential computer has cumulative memory cost *at most* $\mathcal{O}\left(n^2\right)$. This follows because we can only fill $\mathcal{O}(n)$ blocks of memory in $\mathcal{O}(n)$ sequential steps. At best we can hope that the attacker will also need to lock up $\Omega(n)$ blocks of memory for $\Omega(n)$ steps.

[2]In contrast to [3], we use energy cost to refer to bandwidth cost.

blue-moves c_b (representing data movements to/from memory) [30]. Ren and Devadas also proved that the bit reversal graph [25], which forms the core of iMHF candidate Catena-BRG [22], is maximally bandwidth hard in the sense that *any* red-blue pebbling has cost $\Omega(n \cdot c_b)$. However, Ren and Devadas [30] did not prove that any attacker in the parallel random oracle model (pROM) can be viewed as a red-blue pebbling so it was not clear whether or not a graph (e.g., Catena-BRG [22]) with high red-blue pebbling cost is *necessarily* bandwidth hard in the pROM model. Similarly, Ren and Devadas [30] showed that `scrypt` is bandwidth hard under a restrictive assumption about the cache-architecture adopted by the attacker i.e., data from RAM can only be retrieved in large chunks. Prior to our work nothing was known about the bandwidth hardness of key MHF candidates such as PHC winner Argon2i [29, 12], DRSample and aATSample [4].

Our Contributions. We formalize the notion of bandwidth hardness in the parallel random oracle model and show that the bandwidth hardness of an iMHF is indeed captured by the red-blue pebbling game. This does for bandwidth hardness what Alwen and Serbinenko [8] did for CMC when they showed that the CMC of an iMHF is captured by the parallel black pebbling game. In particular, to determine whether a candidate iMHF is sufficiently bandwidth-hard it suffices to analyze the red-blue pebbling costs associated with the corresponding directed-acyclic graph G.

Second, we demonstrate that CMC *lower bounds* can be used to *directly* lower bound energy costs for both iMHFs and dMHFs. Intuitively, an attacker running in time t will pay computation costs at least $t \cdot c_r$, where c_r denotes the energy cost of each random oracle query, and must incur energy cost *at least* $\left(\frac{\mathsf{CMC}}{t \cdot w} - m\right) c_b$ transferring data between cache/RAM. Here, m denotes the number of w bit words that can be stored in cache and c_b denotes the energy costs associated with transferring a w-bit word between cache/RAM — we typically expect that $m \ll n$ and $c_b \gg c_r$. Based on this observation we can show that the energy costs of any attacker with cache size m are at least $\Omega\left(\sqrt{c_b c_r \cdot \mathsf{CMC}/w} - c_b \cdot m\right)$. The result also demonstrates that the goals of high CMC and high bandwidth hardness are well aligned. For example, Alwen *et al.* [6] show that `scrypt` has CMC at least $\Omega(n^2 \cdot w)$ in the parallel random oracle model where the random oracle output is a w-bit word. Combined with our result this implies that `scrypt` has energy cost at least $\Omega\left(\sqrt{c_b c_r} n\right)$ whenever $m = o(n)$. Ren and Devadas [30] had previously shown that `scrypt` has energy cost at least $\Omega(n c_b)$ whenever $m = o(n)$ under a somewhat restrictive assumption about the cache-architecture. While the $\Omega\left(\sqrt{c_b c_r} n\right)$ lower bound on `scrypt` is not tight, it is interesting in that it is unconditional and follows directly from the observation that CMC at least $\Omega(n^2 \cdot w)$.

Third, we establish the first unconditionally tight lower bound on the energy cost of `scrypt`. In particular, we show that in the parallel random oracle model, any algorithm evaluating `scrypt` has energy cost $\Omega(n \cdot c_b)$, by modifying and extending ideas from the reduction of Alwen *et al.* [6]. By contrast, the conditional lower bound of [30] makes a restrictive assumption about the cache-architecture so that prior results of Alwen *et al.* [6] can be used as a blackbox.

Fourth, we introduce a new technique to lower-bound the red-blue pebbling cost of a DAG and we use this new technical hammer to lower-bound the reb-blue pebbling cost of several important iMHF candidates including: Argon2iB (the current version of PHC winner Argon2i [12]), Argon2iA (an older version of Argon2, which is similar to balloon hashing [17]), DRSample and aATSample. For each of these functions we show that if $m = \mathcal{O}\left(n^{1-\epsilon}\right)$ then then any pROM attacker with cache-size $m \cdot w$ bits (m words) *must* incur energy cost *at least* $\min\{\Omega\left(n \cdot c_b\right), \omega(n \cdot c_r)\}$ where the specific $\omega(n \cdot c_r)$ cost term can vary depending on m and the particular iMHF. In an asymptotic sense, we can say that the functions are maximally bandwidth hard as the $\omega(n \cdot c_r)$ cost term will eventually dominate as n grows large so that our lower bound becomes $\Omega\left(n \cdot c_b\right)$. We prove an even stronger lower bound for aATSample. In particular, aATSample is maximally bandwidth hard as long as $m = \mathcal{O}\left(n/\log n\right)$ i.e., any pROM attacker with cache-size m *must* incur energy cost *at least* $\min\{\Omega\left(n \cdot c_b\right), \omega(n \cdot c_r)\}$.

Interestingly, DRSample and aATSample have asymptotically higher CMC as well, which is consistent with our observation that the goal of designing an MHF with high CMC is well aligned with the goal of designing a maximally bandwidth hard function. On the other hand, Argon2iA and Argon2iB are still maximally bandwidth hard even though their CMC is lower than aATSample or DRSample. Thus, bandwidth-hardness does not necessarily imply high CMC.

While we prove that DRSample, aATSample, Argon2iA and Argon2iB are all maximally bandwidth hard in an asymptotic sense, it would be nice to gain a more precise understanding of the constant factors in these bounds. To this end it would be useful to develop an efficient algorithm to find the minimum cost red-blue pebbling of any DAG G. However, our final result is a negative one. In Appendix D we show that it is NP-Hard to compute the minimum cost red-blue pebbling of a general DAG G. This result does not definitively rule out efficient algorithms to compute (or approximate) the minimum cost red-blue pebbling of specific graphs such as DRSample, aATSample or Argon2iB though any such algorithm would have to be targetted to the specific graph structure.

1.1 Graph Pebbling and iMHFs

An iMHF $f_{G,H}$ is defined by a labeling game over a DAG G and a random oracle $H : \{0,1\}^* \to \{0,1\}^w$. In particular, the label ℓ_v of an intermediate node v is computed as $\ell_v = H\left(v, \ell_{v_1}, \ldots, \ell_{v_{\mathsf{indeg}}}\right)$ where $v_1, \ldots, v_{\mathsf{indeg}}$ are the parents of node v in G. The output of the function is the label of the final sink node. Before we provide an overview of our technical results it is necessary to first (informally) introduce the black pebbling game and the red-blue pebbling game.

Black Pebbling. Given a directed acyclic graph (DAG) $G = (V, E)$, the goal of the (parallel) black pebbling game is to place pebbles on all sink nodes of G (not necessarily simultaneously). The game is played in rounds and we use $P_i \subseteq V$ to denote the set of currently pebbled nodes on round i. Initially all nodes are unpebbled, $P_0 = \emptyset$, and in each round $i \geq 1$ we may only include $v \in P_i$ if all of v's parents were pebbled in the previous configuration ($\mathsf{parents}(v) \subseteq P_{i-1}$) or if v was already pebbled in the last round ($v \in P_{i-1}$). More formally, a pebbling sequence $P_0, \ldots, P_t \subseteq V$ is a legal partial pebbling of G if for all pebbling rounds $i \leq t$ we have $\bigcup_{v \in P_i \setminus P_{i-1}} \mathsf{parents}(v) \subseteq P_{i-1}$. If we additionally have $P_0 = \emptyset$ (i.e., we start with no pebbles) and $V \subseteq \bigcup_{i \leq t} P_i$ (i.e., all nodes are pebbled at some point), we say that the pebbling sequence is complete or, simply, a legal pebbling of G. In the sequential pebbling game we can place at most one new pebble on the graph in any round (i.e., we additionally require that $|P_i \backslash P_{i-1}| \leq 1$ for each round $i \leq t$), but in the parallel pebbling game no such restriction applies. We use $\mathcal{P}^{\parallel}(G)$ to denote the set of all legal (complete) black pebblings of the DAG G and we use $\mathcal{P}(G)$ to denote the set of all legal (complete) sequential pebblings of G. Observe that $\mathcal{P}(G) \subset \mathcal{P}^{\parallel}(G)$ since any sequential black pebbling is also a legal parallel pebbling. The space cost of the pebbling is defined to be $\max_i |P_i|$, which intuitively corresponds to minimizing the maximum space required during computation of the associated function, and relates to the space-complexity of the black-pebbling game. Gilbert *et al.* [23] studied the space-complexity of the black-pebbling game and showed that this problem is $\mathsf{PSPACE}-\mathsf{Complete}$ by reducing from the truly quantified boolean formula (TQBF) problem. Given a (partial) black pebbling $P = (P_1, \ldots, P_t)$ of a DAG G, we define the cumulative cost to be $\mathsf{cc}(P) := |P_1| + \ldots + |P_t|$. Then we define $\Pi_{cc}(G) := \min_{P \in \mathcal{P}(G)} \mathsf{cc}(P)$ (resp. $\Pi^{\parallel}_{cc}(G) := \min_{P \in \mathcal{P}^{\parallel}(G)} \mathsf{cc}(P)$) as the minimum cumulative cost of any legal sequential (resp. parallel) black pebbling of G.

Pebbling Reduction in the pROM Model. Alwen and Serbinenko [8] show that under the parallel random oracle model (pROM) of computation, the cryptanalysis of an iMHF, under the amortized time-space metric, can be approximately reduced to the cumulative cost of a pebbling strategy. The result is significant in that it allows future cryptanalysis of iMHF candidates to focus on understanding the (parallel) black pebbling costs of the underlying DAG. In particular, a lower bound on the aAT complexity of the best pebbling for a DAG G immediately yields a lower bound on the aAT complexity of *any* pROM attacker evaluating the function $f_{G,H}$. Intuitively, this means that if G has sufficiently high (parallel) black pebbling cost then it will be prohibitively expensive for an offline attacker to obtain enough hardware to compute the function $f_{G,H}$ at a high rate e.g., millions/billions of times per second.

Red-Blue Pebbling. Given a DAG $G = (V, E)$, the goal of the red-blue pebbling game [24] is again to place pebbles on all sink nodes of G (not necessarily simultaneously) from a starting configuration that contains no pebbles on any nodes. The game is again played in rounds, with each node possibly containing a blue pebble or a red pebble at each time step. Informally, at each time step, for any node v we can swap between a red pebble at v and a blue pebble at v (and vice versa). Each swap is called a blue move, and while there is no limit to the number of blue moves at a single time step, they each have an associated cost c_b. Simultaneously, we may place a red pebble at a node v if all of v's parents contained red pebbles in the previous configuration. This manner of placing a new red pebble is a red move and each occurrence incurs cost c_r. We are allowed to have *at most* m (cache-size) red-pebbles on the graph at any point in time. In a sequential red-blue pebbling we are allowed to place at most one new red pebble on the graph during each round, while no such constraint applies to a parallel red-blue pebbling. Finally, there is a parameter m that denotes a upper bound on the number of nodes that can contain red pebbles at each time step. The total cost of the red-blue pebbling is the sum of the costs induced by the blue moves and the red moves. We define $\mathsf{rbpeb}^{\parallel}(G, m)$ (resp. $\mathsf{rbpeb}(G, m)$) to be the minimum cost of *any* legal parallel (resp. sequential) red-blue pebbling of G that places at most m red-pebbles on the graph at any point in time. We will focus on lower bounding $\mathsf{rbpeb}^{\parallel}(G, m)$ since this also lower bounds the sequential pebbling cost i.e., $\mathsf{rbpeb}(G, m) \geq \mathsf{rbpeb}^{\parallel}(G, m)$. In contrast to black pebbling it will turn out that the difference between sequential/parallel red-blue pebbling costs is minimal i.e., we can show that $\mathsf{rbpeb}(G, m) \geq \mathsf{rbpeb}^{\parallel}(G, m) \geq \mathsf{rbpeb}(G, 2m)$.

1.2 Overview of Our Results

Proving that the Red-Blue Pebbling Game Captures Bandwidth Hardness of iMHFs. We consider the variant of the red-blue pebble game proposed by Ren and Devadas [30] in which red moves have cost c_r and blue moves have cost c_b — note that if $c_r = 0$ then we recover the traditional goal of minimizing the number of cache misses. Ren and Devadas [30] proposed the adoption of red-blue pebbling to model the bandwidth-complexity of iMHFs, with the idea that red moves correspond to hash computations and blue moves correspond to (more expensive) swaps between cache and memory. However, they did not *prove* any connection between red-blue pebbling costs and the *actual* bandwidth-costs of a pROM attacker.

Our contributions are two-fold. First, we formalize the notion of energy cost of a function $f_{G,H}$ in the parallel random oracle model. Second, we prove that $\mathsf{ecost}\left(f_{G,H}\right)$ the energy cost of $f_{G,H}$ is closely related to red-blue pebbling costs. In particular, we prove that any pROM machine computing $f_{G,H}$ with cache-size mw-bits has energy costs $\Omega(\mathsf{rbpeb}^{\parallel}(G, 9m))$. This resolves an open question of [30], and shows that future cryptanalysis of the *bandwidth hardness* of iMHF candidates can focus on the red-blue pebbling cost of the underlying DAG G.

Theorem 1.1. *(Informal, see Theorem 3.3.) $f_{G,H}$ has energy cost at least* $\mathsf{ecost}\left(f_{G,H}, mw\right) \in \Omega\left(\mathsf{rbpeb}^{\parallel}(G, 9m)\right)$.

While Theorem 3.3 is similar to a result of Alwen and Serbinenko who showed that the cumulative memory complexity of $f_{G,H}$ is captured by the black pebbling game [8], we stress that there are several unique challenges in our reduction. Essentially, the pebbling reduction of [8] extracts a black pebbling from the execution trace of a pROM attacker by examining the random oracle queries made during each round i.e., each new pebble that is placed on the graph during round i corresponds directly to a random oracle query that was made during the previous round. To complete the argument Alwen and Serbinenko then use a compression argument to relate the number of pebbles on the graph to the size of the pROM attacker's state during each round. In our setting we need to additionally determine which pebbles are red and blue during each round and we need to relate the number of blue moves to the number of bits transferred to/from memory. However, in the red-blue pebbling model only red moves correspond to random oracle queries. Intuitively, we expect that blue moves correspond to labels that are transferred to/from memory, but an attacker may encode each of these labels in an unexpected way (e.g., encryption). Thus, even if we can observe the data values being transferred to/from memory we stress that we *cannot directly infer* which labels are being transferred making it difficult to extract a legal red-blue pebbling from the execution trace.

We overcome this difficulty by allowing the red-blue pebbling to use a little bit of extra memory (e.g., if the pROM attacker has $m \cdot w$ bits of cache then the red-blue pebbling is allowed to use $9m$ red-pebbles) and by introducing the notion of a *red-blue extension pebbling* of a legal (partial) black pebbling $P = (P_1, \ldots, P_t)$.

In particular, we show that we can partition time into intervals $(t_0 = 0, t_1], (t_1, t_2], \ldots (t_{k-1}, t_k = t]$ in such a way that 1) the pROM attacker transfers *at least* mw bits between cache/memory during each interval $(t_i, t_{i+1}]$, 2) our red-blue extension pebbling uses at most $9m$ red pebbles and, on average, makes at most $\mathcal{O}(m)$ blue moves during each interval $(t_i, t_{i+1}]$. Thus, the pROM attacker incurs cost at least $\Omega(km \cdot c_b)$ transferring data between cache and memory while the red-blue extension pebbling has cost at most $\mathcal{O}(kmc_b)$ for blue moves.

To partition time into intervals we introduce a set $\mathbf{QueryFirst}(x, y)$ that intuitively corresponds to the data-labels that *appear first as input* to a random oracle query during the time interval $[x, y]$ *before* the label appears as the output of some random oracle query during the time interval $(x, y]$. We then define t_1 to be the minimum pebbling round such that there exists $1 < j_1 \le t_1$ such that $\mathbf{QueryFirst}(j_1, t_1)$ has size at least $3m$. Similarly, once $t_1 < \ldots < t_{i-1}$ have been defined we can define $t_i > t_{i-1}$ to be the minimum pebbling round such that there exists $t_{i-1} < j_i \le t_i$ s.t. $\mathbf{QueryFirst}(j_i, t_i)$ has size at least $3m$. At the beginning of each interval $(t_i, t_{i+1}]$ our red-blue extension pebbling will place red pebbles on all nodes in the set $\mathbf{QueryFirst}(t_i, t_{i+1})$ (i.e., to "load" these values into cache). We can argue that there are *at most* $4m$ pebbles in this set $\mathbf{QueryFirst}(t_i, t_{i+1})$. Thus, we can accomplish this initial step legally since the extension pebbling is allowed to use up to $9m > 4m$ red-pebbles. Once we have red pebbles placed on all of these nodes the extension pebbling is able to finish this interval *without* changing any other blue nodes into red-nodes (i.e., zero cache misses). In particular, during the remainder of the interval we will simply assign every newly pebbled node to have the color red. To ensure that we don't use too many red pebbles during each intermediate round $t_i < j \le t_{i+1}$ we can discard our red pebble on node v if this pebble will never be needed to repebble any of v's parents during the current time interval or if v will be (re)pebbled before any of its parents (if we need node v for a future interval $(t_{i'}, t_{i'+1}]$ with $i' > i$ then we can convert node v to a blue pebble and "charge" this cost to the future time interval). Thus, we can upper bound the total number of red pebbles as $m + |\mathbf{QueryFirst}(t_i, t_{i+1})| + \max_{t_i \le j \le t_{i+1}} |\mathbf{QueryFirst}(j, t_{i+1})| \le m + 4m + 4m = 9m$. Intuitively, $|\mathbf{QueryFirst}(t_i, t_{i+1})| \le 4m$ accounts for red-pebbles added at the beginning of the interval, $|\mathbf{QueryFirst}(j, t_{i+1})| \le 4m$ upper bounds the number of additional red-pebbles that need to be kept around and m upper bounds the number of new red-pebbles placed on the graph in each round. Finally, at the end of the interval we can use *at most* $O(m)$ blue moves to free cache by converting any of our current $9m$ red pebbles to blue pebbles i.e., if these pebbles will be required for future time intervals.

To prove that the pROM attacker must transfer *at least* mw bits from memory during each interval we rely on an extractor argument. In particular, let γ_i encode the messages transferred to/from cache during the interval $(t_i, t_{i+1}]$. Our extractor will extract $3m$ labels (without querying the random oracle at these points) by simulating the pROM attacker starting with a hint. The labels we will extract correspond to the nodes in the set $\mathbf{QueryFirst}(j_{i+1}, t_{i+1})$ of size $|\mathbf{QueryFirst}(j_{i+1}, t_{i+1})| \ge 3m$ where $j_{i+1} \in (t_i, t_{i+1}]$. The hint consists of γ_i along with other information such as the current state of the cache (at most mw bits), indices of the labels that we want to extract (at most $|\mathbf{QueryFirst}(t_i, t_{i+1}))| \log n \le 4m \log n$ bits to encode), and the index of the first query in which each label appears as input to a random oracle query (at most $4m \log q$ bits to encode where q is an upper bound on number of queries made by the attacker). Since a random oracle is incompressible, the extractor's hint must have length at least $3mw$ if we expect the extractor to output at least $3m$ labels (i.e., at least $3m$ distinct random oracle outputs of length w assuming there are no hash collisions) without querying the random oracle at these points so it follows that $|\gamma_i| \ge m \cdot w$.

On the Bandwidth Hardness of Important iMHF Candidates. In Section 5, we provide lower bounds on the bandwidth hardness of several important iMHF candidates including Argon2iA, Argon2iB [12], $\mathsf{aATSample}$ and $\mathsf{DRSample}$ [4]. We use Argon2iA to refer to v1.1 and we use Argon2iB to refer to versions v1.2+ [3]. Thus, Argon2iB (the current version of Argon2i) is particularly important to cryptanalyze as it won the password hashing competition [29] and is being considered for standardization by the Cryptography Form Research Group (CFRG) of the IRTF [13].$\mathsf{aATSample}$ and $\mathsf{DRSample}$ are important to study as they

[3]The specification of Argon2i has changed several times, but the only changes that affect our analysis are changes that affect the underlying DAG G. A change to the edge distribution was introduced in v1.2 where a non-uniform indexing was introduced. We use Argon2iB to refer to the version that is currently being considered for standardization by the Cryptography Form Research Group (CFRG) of the IRTF[13].

are the first *practical* iMHF candidate with nearly asymptotically optimal cmc[4].

For context we observe that there is always red-blue pebbling strategy that makes at most $\mathcal{O}(n)$ blue moves and at most $\mathcal{O}(n)$ red moves for a total cost of *at most* $\mathcal{O}(nc_b + nc_r)$. In particular, the naive pebbling strategy simply pebbles nodes in topological order immediately converting red nodes to blue nodes whenever we need to free up cache and converting blue nodes back to red nodes only when needed. This naive strategy works for any cache size m as long as m is larger than the indegree of the graph — if m is smaller than the indegree then there is no legal red-blue pebbling. If $m \geq n$ then cache is large enough to store *all* labels there is a naive red-blue pebbling strategy that makes at most $\mathcal{O}(n)$ red-moves and 0 blue moves. For the families of graphs generated by aATSample and DRSample [4] we show the following:

Theorem 1.2. *Let G be a graph generated by* aATSample*. Then there exists constants $C, C' > 0$ so that for all $m \leq \frac{Cn}{\log n}$,*

$$\mathsf{rbpeb}^{\parallel}(G, m) \geq C' \cdot \min(n \cdot c_b, (n \log n)c_r),$$

holds with high probability.

Our lower bound for DRSample requires the slightly stronger (but still realistic) assumption that $m \leq C'n^{\rho}$ for some constant $\rho < 1$ as opposed to the slightly weaker assumption that $m \leq \frac{Cn}{\log n}$ in Theorem 1.2.On the positive side the red cost term $\Omega(n^{3/2-\rho/2})c_r$ from Theorem 1.3 is an improvement over Theorem 1.2. We typically expect that $n^{3/2-\rho/2}c_r \geq nc_b$ in which case the lower bound from Theorem 1.3 is simply $\Omega(n \cdot c_b)$. Because the first $n/2$ nodes from aATSample form a copy of DRSample the same asymptotic lower bound applies when $m \leq C'n^{\rho}$.

Theorem 1.3. *Let G be a graph generated by* DRSample *or* aATSample *and $0 < \rho < 1$. Then there exists constants $C, C' > 0$ so that for all $m \leq C'n^{\rho}$, with high probability,*

$$\mathsf{rbpeb}^{\parallel}(G, m) \geq C \cdot \min\left(n \cdot c_b, n^{3/2-\rho/2} \cdot c_r\right).$$

Our lower bounds for Argon2iA and Argon2iB are comparable to DRSample. In fact, the red cost term is slightly better than in Theorem 1.3 particularly when m is small. For example, if $\epsilon = 0.9$ then $m = n^{1/10}$ (unrealistically small in practice) and the red-cost term in Theorem 1.4 is $n^{1+\epsilon}c_r = n^{1.9}r$. By contrast, if $m = n^{0.9}$ in Theorem 1.3 the red-cost term is just $n^{1.05}c_r$.

Theorem 1.4. *Let G be a random Argon2iB (resp. Argon2iA) graph. Then there exists constants $C, C' > 0$ so that for any $0 < \epsilon < 1$ and for all $m \leq C'n^{1-\epsilon}$, with high probability,*

$$\mathsf{rbpeb}^{\parallel}(G, m) \geq C \cdot \min(nc_b, n^{1+\epsilon}c_r).$$

At a technical level our template to establish each of these lower bounds is similar. We show that the graph is "well dispersed." Essentially, if our block size is b, then we show that for every interval $I = [i, j] \subseteq [n/2, n]$ of $\Omega(n/b)$ nodes in the second half and *almost every* block B of b consecutive nodes in the first half $[n/2]$ there is an edge from some node in the second half of B to some node in I [5]. We then consider the pebbling interval $[t_i, t_j]$ beginning at the time t_i during which a pebble is first placed on node i and ending at the time t_j during which a pebble is first placed on node j. If block B initially contains no red-pebble and there is an edge from the second half of B to I then either 1) we will need to make a blue move to place a red-pebble on block B or 2) we will need to make at least $b/2$ red-moves to repebble all of the nodes in the first half of B. If the cache size is $m \in o(n/b)$ then most of these $\Omega(n/b)$ blocks will begin with no pebbles in cache. Because

[4]Prior work [5] gave a theoretical construction of an iMHF with $\Omega\left(\frac{n^2 \cdot w}{\log n}\right)$ (matching DRSample and aATSample), but to the best of our knowledge no implementation exists. By contrast, DRSample can be easily implemented by modifying the source code for Argon2iB and these modifications do not adversely impact performance [4]. *Any* iMHF $f_{G,H}$ has cmc at most $\mathsf{cmc}(f_{G,H}) \in \mathcal{O}\left(\frac{n^2 \cdot w \cdot \log\log n}{\log n}\right)$ [2, 3] so $\mathsf{cmc}(\mathsf{DRSample}) \in \Omega\left(\frac{n^2 \cdot w}{\log n}\right)$ and $\mathsf{cmc}(\mathsf{aATSample}) \in \Omega\left(\frac{n^2 \cdot w}{\log n}\right)$ [4] are essentially tight.

[5]For DRSample it suffices to show that this property holds for sufficiently many blocks B.

the graph is "well dispersed" we will need to place a red pebble on *at least one node* from almost every block. Thus, during the interval $[t_i, t_j]$, it is either the case that 1) we make $\Omega(n/b)$ blue moves, or 2) we make $\Omega(n)$ red moves. The total cost can be lower bounded by summing over all $(n/2)/|I|$ such time intervals.

On the Relationship between Bandwidth Complexity and Cumulative Memory Complexity. We show that bandwidth complexity and cumulative memory complexity are intricately related concepts. If $\mathsf{rbpeb}^{\parallel}(G, m)$ is the minimum energy cost of *any* legal *parallel* reb-blue pebbling of G with cache size m and Π_{cc} is the cumulative complexity of *sequential* black pebbling, then

Theorem 1.5.

$$\mathsf{rbpeb}^{\parallel}(G, m) \geq \min_t \left(2c_b \left(\frac{\Pi_{cc}(G)}{t} - 2m \right) + c_r t \right) \in \Omega\left(\sqrt{c_b \cdot c_r \cdot \Pi_{cc}(G)} \right),$$

where m is the cache size, t is the number of steps in the pebbling, c_b is the cost of a blue move and c_r is the cost of a red move.

Theorem 1.5 demonstrates that the goals of designing an MHF with high cumulative complexity and high bandwidth complexity are well aligned. In fact, we use Theorem 1.5 to show that a family $\{G_n\}_{n=1}^{\infty}$ of constant indegree DAGs constructed by Schnitger [31] has high energy costs *because* the *sequential* black pebbling cost is $\Pi_{cc}(G_n) \in \Omega(n^2)$ [7]. In particular, the optimal red-blue pebbling must either make $t = n\sqrt{c_b/c_r}$ red-moves or the pebbling strategy will use at least $(n\sqrt{c_r}c_b - 2m)$ blue moves. As an intermediate step to proving Theorem 1.5 we show that $\mathsf{rbpeb}^{\parallel}(G, m) \geq \mathsf{rbpeb}(G, 2m)$. This result is interesting as it suggests that an attacker will not be able to dramatically decrease energy costs by exploiting parallelism. By contrast, for *any* constant indegree DAG G it is known that the parallel cumulative pebbling cost is at most $\Pi_{cc}^{\parallel}(G) \in \mathcal{O}\left(\frac{n^2 \log\log n}{\log n}\right)$ [2] while it is known that $\Pi_{cc}(G_n) \in \Omega(n^2)$ for the constant indegree DAGs constructed by Schnitger [31].

We also prove a similar theorem that directly relates ecost and cmc. In particular, we show that

$$\mathsf{ecost}\left(f_{G,H}\right) \in \Omega\left(\sqrt{c_b c_r \mathsf{cmc}\left(f_{G,H}\right)} - c_b m \right).$$

Crucially, this bound applies to *any* MHF not just for iMHFs. For iMHFs we could use our pebbling reduction to relate $\mathsf{ecost}\left(f_{G,H}\right)$ to $\mathsf{rbpeb}^{\parallel}(G)$ and we could use [8] to relate $\mathsf{cmc}\left(f_{G,H}\right)$ to $\Pi_{cc}(G)$, but no such pebbling reduction is known for dMHFs. Combining our result with a result of Alwen et al. [6] we obtain the following lower bound for `scrypt`: $\mathsf{ecost}\left(\texttt{scrypt}\right) \in \Omega\left(\sqrt{c_b c_r} n\right)$. While we later obtain a tighter lower bound $\mathsf{ecost}\left(\texttt{scrypt}\right) \in \Omega\left(n \cdot c_b\right)$, the previous result is interesting because it follows *immediately* from the cumulative memory complexity of `scrypt` without additional analysis.

On the Bandwidth Hardness of `scrypt`. In Section 6, we provide a tight lower bound on the bandwidth hardness of `scrypt` [28] eliminating a restrictive assumption required in the lower bound of Ren and Devadas [30]. Our pebbling analysis only applies to iMHFs so we are unable to apply pebbling arguments to lower bound the energy cost of dMHFs such as `scrypt`. In particular, Theorem 1.6 shows that *any* algorithm in the parallel random oracle model making at most $q \leq 2^{w/20}$ queries to the random oracle $H : \{0,1\}^* \to \{0,1\}^w$ and computing `scrypt` correctly with probability at least ϵ has energy cost $\Omega(\epsilon n c_b)$.

Theorem 1.6. *Whenever $4 \log n < w$, $q \leq 2^{w/20}$, $\frac{n}{4m} \cdot c_r > c_b$, and $\epsilon \geq 2(\exp\left(-\frac{n}{8}\right) + \frac{3}{2}n^3 2^{-w} + qn^2 2^{-w} + 2^{-mw/5})$ the following statement holds in the parallel random oracle model:*

$$\mathsf{ecost}_{q,\epsilon}(\texttt{scrypt}_n, m \cdot w) \geq \frac{\epsilon}{2} \cdot \frac{nc_b}{16}$$

Ren and Devadas [30] prove that the energy cost of `scrypt` is $\Omega(nc_b)$ under a restrictive constraint that an adversary must fetch w bits at a time. Under such a restrictive assumption the extractor argument from Alwen et al. [6] can be used as a black box without any modification. In particular, the only way for an adversary to obtain a label is to *either* recompute the label without accessing memory at all or load *at least* w bits of data (one full label) from memory. In our unrestricted setting the attacker has no such restriction and transfers arbitrary bits of data from/to memory at a time e.g., the attacker could choose to only transfer $\sqrt{w}$ bits from memory in an attempt to minimize bandwidth costs. Proving the lower bound $\Omega(nc_b)$ without this constraint is challenging as we cannot simply use the results in Alwen et al. [6] as a black box. We give the first tight *unconditional* lower bound on the bandwidth hardness of `scrypt` in the parallel random oracle model i.e., without the restrictive constraint that an adversary must fetch w bits at a time.

On the Computational Complexity of Minimum Cost Red-Blue Pebbling. While we can establish asymptotic lower bounds on the energy cost of important iMHF candidates, one would ideally want to find the precise energy cost for each function. In particular, given a graph G and a cache parameter m we would like to compute $\mathsf{rbpeb}^{\parallel}(G,m)$ precisely. However, we show in Appendix D that, unfortunately, exactly computing the red-blue pebbling cost of a DAG G is $\mathtt{NP}-\mathtt{Hard}$, even under realistic assumptions about c_b and c_r:

Theorem 1.7 (Informal)**.** *Even for $c_b > 10000c_r$, the problem of determining the red-blue pebbling cost of a directed acyclic graph G is $\mathtt{NP}-\mathtt{Hard}$.*

A result of Demaine and Liu [18, 26] implies that it is PSPACE hard to compute $\mathsf{rbpeb}^{\parallel}(G,m)$ when $c_r = 0$ (computation is free)[6]. However, we stress that *in practice* we have $c_r > 0$ (computation may be *cheap*, but it is not *free*). Furthermore, if we ensure that $c_r > 0$ and $c_b/c_r \leq \mathsf{poly}(n)$ the decision problem $\mathsf{rbpeb}^{\parallel}$ = "is $\mathsf{rbpeb}^{\parallel}(G,m) \leq k$" is in $\mathtt{NP}$[7] so, unless $\mathtt{NP} = \mathsf{PSPACE}$, the decision problem is fundamentally different when computation is not free. While the decision problem $\mathsf{rbpeb}^{\parallel}$ is important for the cryptanalysis of MHFs to the best of our knowledge nothing was known about the complexity of this problem prior to our paper.

Gilbert *et al.* [23] previously showed that the following decision problem was PSPACE complete: Given a DAG G decide if there is a legal black pebbling with space complexity at most m i.e., during every pebbling round there are at most m pebbles on the graph. Gilbert *et al.* showed that the minimum space black pebbling problem was $\mathsf{PSPACE}-\mathsf{Hard}$ by reduction from the Truly Quantified Boolean Formula (TQBF) problem. Observing that *any* $3-\mathsf{SAT}$ instance ϕ with n variables is also a TQBF instance (albeit with no $\forall$ quantifiers) their reduction allows us to transform ϕ into a DAG G_ϕ. The graph G_ϕ has the property that it can be pebbled with at most $m = 3n+3$ black pebbles if and only if ϕ is satisfiable. In Appendix D we detail a gadget to append to G_ϕ to create a graph H_ϕ so that $\mathsf{rbpeb}^{\parallel}(H) = x_1$ if ϕ is a satisfiable assignment, but $\mathsf{rbpeb}^{\parallel}(H_\phi) > x_1$ if ϕ is not a satisfiable assignment.

2 Preliminaries

We use $[n]$ to denote the set $\{1,2,\ldots,n\}$ and $[a,b] = \{a,a+1,\ldots,b\}$ where $a,b \in \mathbb{N}$ with $a \leq b$. Similarly, we use $(a,b]$ to denote the set $[a,b] - \{a\}$.

We assume a given directed acyclic graph (DAG) $G = (V,E)$ is labeled in topological order and when G has *size* n we will use $V = [n]$ to denote the set of vertices $E \subseteq \{(i,j) \;:\; 1 \leq i < j \leq n\}$ denotes the set of all directed edges in G. We say a node $v \in V$ has indegree $\delta = \mathsf{indeg}(v)$ if there exist δ incoming edges $\delta = |(V \times \{v\}) \cap E|$. We say that G has indegree $\delta = \mathsf{indeg}(G)$ if the maximum indegree of any node of G is δ. A node with indegree 0 is called a source node and a node with no outgoing edges is called a sink. We use $\mathsf{parents}_G(v) = \{u \in V : (u,v) \in E\}$ to denote the parents of a node $v \in V$ and similarly for a set $S \subseteq V$, we define $\mathsf{parents}_G(S) = \{u \in V : (u,v) \in E, v \in S\}$. In general, we use $\mathsf{ancestors}_G(v) = \bigcup_{i\geq 1} \mathsf{parents}^i_G(v)$ to denote the set of all ancestors of v — here, $\mathsf{parents}^2_G(v) = \mathsf{parents}_G(\mathsf{parents}_G(v))$ denotes the grandparents of v and $\mathsf{parents}^{i+1}_G(v) = \mathsf{parents}_G\left(\mathsf{parents}^i_G(v)\right)$. When G is clear from context we will simply write $\mathsf{parents}$

[6]In particular, $\mathsf{rbpeb}^{\parallel}(G,m) = 0$ if and only if there is a legal black pebbling of G using at most m black pebbles where the latter decision problem is PSPACE complete [23].

[7]In particular, if $c_r > 0$ and $c_b/c_r = \mathsf{poly}(n)$ we are guaranteed that the optimal red-blue pebbling runs in time at most $\mathsf{poly}(n)$. Thus, yes instances of our decision problem admit a polynomial size witness.

(resp. $\mathsf{ancestors}$). We denote the set of all sinks of G with $\mathsf{sinks}(G) = \{v \in V : \nexists (v,u) \in E\}$, the nodes with no outgoing edges.

We often consider the set of all DAGs of equal size $\mathbb{G}_n = \{G = (V,E) \ : \ |V| = n\}$ and often will bound the maximum indegree $\mathbb{G}_{n,\delta} = \{G \in \mathbb{G}_n : \mathsf{indeg}(G) \le \delta\}$. For directed path $p = (v_1, v_2, \ldots, v_z)$ in G, its length is the number of nodes it traverses, $\mathsf{length}(p) := z$ (as opposed to the number of edges). We say the depth $d = \mathsf{depth}(G)$ of DAG G is the length of the longest directed path in G.

An iMHF can be specified by a DAG G and a random oracle H as in the next definition.

Definition 2.1. *Given a directed acyclic graph $G = (V = [n], E)$ with a set of sink nodes $\mathsf{sinks}(G)$ and a random oracle function $H : \Sigma^* \to \Sigma^\ell$ over an alphabet Σ, we define the labeling of graph G as $\mathsf{lab}_{G,H} : \Sigma^* \to \Sigma^*$. We omit the subscripts G, H when the dependency on the graph G and hash function H is clear. In particular, given an input x the (H, x) labeling of G is defined recursively by*

$$\mathsf{lab}_{H,x}(v) = \begin{cases} H(v, x), & \mathsf{indeg}(v) = 0 \\ H\left(v, \mathsf{lab}_{H,x}(v_1), \ldots, \mathsf{lab}_{H,x}(v_d)\right), & \mathsf{indeg}(v) > 0, \end{cases}$$

where $v_1, \ldots, v_d$ are the parents of v in G, according to some predetermined lexicographical order. It will also be convenient to use $\mathtt{prelab}(v) = (v, \mathsf{lab}_{H,x}(v_1), \ldots, \mathsf{lab}_{H,x}(v_d))$ to denote the prelabel of node v i.e., the random oracle query whose output is $\mathsf{lab}_{H,x}(v)$. We define

$$f_{G,H}(x) = \{\mathsf{lab}_{H,x}(s)\}_{s \in \mathsf{sinks}(G)}.$$

If there is a single sink node s_G then $f_{G,H}(x) = \mathsf{lab}_{H,x}(s_G)$.

We will often consider graphs obtained from other graphs by removing subsets of nodes. Thus if $S \subset V$, then let $G - S$ be the DAG obtained from G by removing nodes S and incident edges.

Given a directed acyclic graph (DAG) $G = (V, E)$ the goal of the red-blue pebbling game is to place pebbles on all sink nodes of G (not necessarily simultaneously).

Let $\mathcal{RB} = ((B_0, R_0), (B_1, R_1), \ldots, (B_t, R_t))$ (resp. $\mathcal{RB}^{\parallel}$) denote the set of all sequential (resp. parallel) red-blue pebblings of a DAG G. The game is played in rounds and we use $B_i \subseteq V$ (resp. $R_i \subseteq V$) to denote the set of nodes with blue pebbles (resp. red pebbles) in round i. Initially, no nodes contain pebbles, so that $B_0 \cup R_0 = \emptyset$. The goal is to eventually place a red-pebble on every node in V (not-necessarily simultaneously) so we require that $V \subseteq \bigcup_i R_i$. We also require that in every round $i > 0$ we have (1) $\mathsf{parents}\left(R_i \setminus (R_{i-1} \cup B_{i-1})\right) \subseteq R_{i-1}$, (2) $B_i \setminus B_{i-1} \subseteq R_{i-1}$ and (3) $|R_i| \le m$.

We let $\mathcal{RB}^{\parallel}(G, m)$ be the set of all valid parallel red-blue pebblings of G with a cache-size of m pebbles. Intuitively, in each round $i \ge 1$ we may place a red pebble on a node $v \in V$ if either $\mathsf{parents}(v) \subseteq R_{i-1}$ all of v's parents contain red pebbles in the previous configuration (called a *red move*) or v contained a blue pebble in the previous round (called a *blue move*). On the other hand, we may place a blue pebble at $v \in P_i$ (also called a *blue move*) if v contained a red pebble in the previous round. Blue moves represent data transfer to/from memory and are more expensive than red-moves (computation).

We say that a pebbling $((B_0, R_0), (B_1, R_1), \ldots, (B_t, R_t))$ is sequential if $|R_i \setminus R_{i-1}| \le 1$ for all $0 < i \le t$, while for a parallel pebbling we make no such restriction. Note that $\mathcal{RB} \subseteq \mathcal{RB}^{\parallel}$ since any sequential pebbling is a legal parallel pebbling.

Formally we define a legal (partial) red-blue pebbling as below:

Definition 2.2. *A pebbling $((B_0, R_0), (B_1, R_1), \ldots, (B_t, R_t))$ is a legal partial red-blue pebbling of G with a cache size of m pebbles if for all $0 < i \le t$ we have: (1) $|R_i| \le m$, (2) $\mathsf{parents}\left(R_i \setminus (R_{i-1} \cup B_{i-1})\right) \subseteq R_{i-1}$, (3) $B_i \setminus B_{i-1} \subseteq R_{i-1}$, (4) $B_0 = R_0 = \emptyset$, (5) (for sequential pebbling only) $|R_i \setminus R_{i-1}| \le 1$. Furthermore, the pebbling is also complete (i.e. a legal red-blue pebbling of G) if (6) $\mathsf{sinks}(G) \subseteq \cup_{i=1}^t R_i$.*

Let $\#BM_i$ and $\#RM_i$ denote the number of blue moves and the number of red moves, respectively, during round i.[8] Formally,

$$\#BM_i = |\{v \in R_i \setminus R_{i-1} \ : \ \mathsf{parents}(v) \not\subset R_{i-1}\}| + |B_i \setminus B_{i-1}|$$
$$\#RM_i = |R_i \setminus R_{i-1}| - |\{v \in R_i \setminus R_{i-1} \ : \ \mathsf{parents}(v) \not\subset R_{i-1}\}|$$

Given cost parameters c_r and c_b, we define the energy cost of a red-blue pebbling $(R, B) = ((R_1, B_1), \ldots, (R_t, B_t))$ to be

$$\mathsf{rbpeb}^{\parallel}\left((R, B)\right) = \sum_{i=1}^{t} c_b \#BM_i + c_r \#RM_i \ .$$

Generally, we assume c_b is much larger than c_r. Finally, we define

$$\mathsf{rbpeb}^{\parallel}\left(G, m\right) = \min_{(R,B)\in\mathcal{RB}^{\parallel}(G,m)} \mathsf{rbpeb}^{\parallel}\left((R, B)\right)$$

to be the cost of the optimal red-blue pebbling of G with maximum cache-size of m red pebbles.

2.1 Depth-Robustness

Definition 2.3 (Block Depth-Robustness)**.** *Given a node v, let $N(v, b) = \{v-b+1, \ldots, v\}$ denote a segment of b consecutive nodes ending at v. Similarly, given a set $S \subseteq V$, let $N(S, b) = \cup_{v\in S} N(v, b)$. We say that a DAG G is (e, d, b)-block-depth-robust if for every set $S \subseteq V$ of size $|S| \leq e$, we have $\mathsf{depth}(G - N(S, b)) \geq d$. If $b = 1$, we simply say G is (e, d)-depth-robust and if G is not (e, d)-depth-robust, we say that G is (e, d)-depth-reducible.*

Note that when $b > 1$, (e, d, b)-block-depth robustness is a strictly stronger notion than (e, d)-depth-robustness since for any set S with $|S| \leq e$ it follows that $N(S, 1) \subset N(S, b)$. Hence, $(e, d, b \geq 1)$-block depth robustness implies (e, d)-depth robustness. On the other hand, (e, d)-depth robustness only implies $(e/b, d, b)$-block depth robustness.

The cumulative memory complexity of an iMHF is very closely related to the notion of depth-robustness [2, 5, 4, 16]. In particular, we know that $\Pi^{\parallel}_{cc}(G) \geq ed$ [5] for any (e, d)-depth-robust DAG and that $\Pi^{\parallel}_{cc}(G) \in \mathcal{O}\left(en + n \cdot \sqrt{dn}\right)$ for any graph that is not (e, d)-depth robust [2]. We will show that $\Pi^{\parallel}_{cc}(G)$ can be used to lower bound $\mathsf{rbpeb}^{\parallel}(G, m)$, thus depth-robustness can also be a useful tool in bandwidth hardness. For DAGS that contain edges $(i, i + 1)$ for each $i < n$ (all of the DAGs we consider) one can occasionally use block depth robustness to prove tighter bounds e.g., [4, 16].

3 Modeling Energy Complexity as Red-Blue Pebbling

In this section we show that the energy cost of the function $f_{G,H}$ is characterized by the reb-blue pebbling cost $\mathsf{rbpeb}^{\parallel}(G, m)$ in the parallel random oracle model just as Alwen and Serbinenko [8] showed that cumulative memory complexity can be characterized by the black pebbling game. Similar to [8] our reduction uses Lemma 3.1 as a core building block. In particular, if the energy cost is significantly smaller than $\mathsf{rbpeb}^{\parallel}(G, 9m)$ for a pROM attacker with $m \cdot w$ bits of cache then we can build an extractor that receives a small hint and predicts the random oracle output on a larger set of indices contradicting Lemma 3.1. One of the unique challenges we face when designing our extractor is that it is not obvious how to relate messages between cache and main memory to specific blue pebbling moves. By contrast, a black pebbling move always corresponds to a specific random oracle query.

[8]In some cases we may have $v \in B_{i-1}$ and $\mathsf{parents}(v) \subset R_{i-1}$ so that we could place a red pebble on node v using either a red move or a blue move. In such cases we will assume that this is accomplished by a red move, since blue moves will be more expensive.

Lemma 3.1. *[21] Let* $\mathtt{HINT}$ *be a set of hints that can be given,* B *be a series of random bits and* $\mathcal{A}$ *be an algorithm that receives as input some hint* $\mathtt{hint} \in \mathtt{HINT}$ *and can adaptively query* B *at specific indices. Let* $\mathbf{WIN}_{\mathcal{A},\mathtt{hint}}$ *denote the event that* $\mathcal{A}$*, given* $\mathtt{hint} \in \mathtt{HINT}$ *as input, eventually outputs a subset of* k *indices* $i_1, \ldots, i_k$ *that were not previously queried as well as the corresponding values* $B[i_1], \ldots, B[i_k]$ *of each bit then*

$$\Pr\left[\exists \mathtt{hint} \in \mathtt{HINT}.\ \mathbf{WIN}_{\mathcal{A},\mathtt{hint}}\right] \leq \frac{|\mathtt{HINT}|}{2^k} ,$$

where the randomness is taken over the selection of B*.*

3.1 Memory and Cache in the Parallel Random Oracle Model

Before we present our reduction it is first necessary to give a formal definition of energy costs in the pROM model.

We define a state of an algorithm $\mathcal{A}^{H(\cdot)}$ to be the tuple (σ, ξ), where σ contains the contents of the cache and has size at most mw bits, and ξ contains the contents of the memory. We consider a pROM attacker $\mathcal{A}^{H(\cdot)}$ with cache size $m \cdot w$ who is given oracle access to a random oracle $H : \{0,1\}^* \to \{0,1\}^w$. In particular, the cache is large enough to store m labels. An execution of $\mathcal{A}^{H(\cdot)}$ on input x proceeds in rounds as follows. Initially, the state at time 0 is (σ_0, ξ_0) where ξ_0 is empty and σ_0 encodes the initial input x. At the beginning of round i the attacker is given the initial state $(\sigma_{i-1}, \xi_{i-1})$ as well as the answers A_{i-1} to any random oracle queries that were asked at the end of the last round. The algorithm $\mathcal{A}^{H(\cdot)}$ may then perform arbitrary computation and/or transfer data between memory and cache. The round ends when the attacker outputs a new state (σ_i, ξ_i) along with a batch of queries $Q_i = \{q_1^i, q_2^i, \ldots, q_{k_i}^i\}$. Since the attacker only has cache-size $m \cdot w$ we only allow the attacker to make *at most* $|Q_i| \leq m$ queries during a single step (otherwise the attacker won't even have room to store all of the random oracle responses in cache). In particular, we require that $|\sigma_i| + k_i w \leq mw$ where $k_i = |Q_i|$ denotes the number of random oracle answers given to $\mathcal{A}^{H(\cdot)}$ at the beginning of round i. Similarly, we require that for all rounds i we have $\sum_{j=1}^{k_i} \left|q_j^i\right| \leq mw$ (we must have enough room in cache to store the random oracle queries).

We allow the attacker to specify *arbitrary* functions F_1, F_2, F_3 and F_4 to model communication between cache and memory and subsequent state updates during each round so long as the specification of each function is independent of the random oracle H (e.g., we cannot query the random oracle in between rounds). In particular, the function $F_1\left(\sigma_{i-1}, A_{i-1}\right) = r_i^1$ is used to specify the *first* message we will send to memory during round i — in the event that we don't send any message to memory we define $F_1\left(\sigma_{i-1}, A_{i-1}\right) = \perp$. Similarly, the function $F_2(\xi_{i-1}, r_i^1) = s_i^1$ specifies the response from memory (or $\perp$ if there is no response). Once $r_i^1, s_i^1, \ldots, r_i^{j-1}, s_i^{j-1}$ have been defined we set

$$r_i^j = F_1\left(\sigma_{i-1}, A_{i-1}, r_i^1, s_i^1, \ldots, r_i^{j-1}, s_i^{j-1}\right),$$
$$s_i^j = F_2\left(\xi_{i-1}, r_i^1, s_i^1, \ldots, r_i^{j-1}, s_i^{j-1}, r_i^j\right).$$

We terminate when $r_i^j = \perp$ or when $s_i^j = \perp$.

We let $R_i = \{r_i^1, r_i^2, \ldots, r_i^{\ell_i}\}$ denote the sequence of messages sent from cache to memory[9] during round i and we let $S_i = \{s_i^1, s_i^2, \ldots, s_i^{\ell_i}\}$ denote the responses sent from memory back to the cache. Finally, the round ends when the attacker uses the function $F_3\left(\xi_{i-1}, R_i, S_i\right) = \xi_i$ to output a new state ξ_i for memory and $F_4\left(\sigma_{i-1}, R_i, S_i\right)$ to output a new state σ_i for cache and a new batch Q_i of at most m random oracle queries. At this point $\mathcal{A}^{H(\cdot)}$ outputs the next state (σ_i, ξ_i) along with the next batch of queries Q_i

Crucially, the functions F_2 and F_3, which are used to generate response from main memory and update the state of main memory at the end of the round, do not have access to σ_{i-1} (the state of cache) or A_{i-1} (the answers to random oracle queries). In particular, any information about σ_{i-1} (cache-state) and A_{i-1} (most recent answers to random oracle queries) that main memory receives must be communicated through one of the messages in the set R_i. Similarly, the functions F_1 and F_4 are used to generate the requests sent

[9]In this subsection we use R_i to denote messages from cache to memory instead of a pebbling configuration.

from cache to main memory, to update the state of cache σ_i at the end of the round and to output the next batch Q_i of random oracle queries. Crucially these functions do not have access to ξ_{i-1} (the state of memory). Thus, any information about ξ_{i-1} must be communicated through one of the responses in the set S_i.

Dziembowski *et al.* [20] also addresses communication between two parties, A_{small} (e.g., a space-bounded virus) and A_{big}, over a bounded channel. However, both parties in this model can query the random oracle. This is a crucial difference, since one of the parties in our model, the main memory, is strictly forbidden from querying the random oracle to avoid trivialization of the problem (e.g., the attacker can perform all computation in RAM with no blue moves).

Execution Trace. We define the execution trace of the algorithm $\mathcal{A}^{H(\cdot)}$ by the sequence of cache states, memory states, messages passed between cache and memory, and queries made to the random oracle H. Formally, the execution trace is $\mathsf{Trace}_{\mathcal{A},R,H}(x) = \{(\sigma_i, \xi_i, R_i, S_i, Q_i)\}_{i=1}^t$, where the trace $\mathsf{Trace}_{\mathcal{A},R,H}(x)$ is dependent on the algorithm $\mathcal{A}^{H(\cdot)}$, random oracle H, internal randomness R, and input value x. Given $S_i = \{s_i^1, s_i^2, \ldots, s_i^{\ell_i}\}$ and $R_i = \{r_i^1, r_i^2, \ldots, r_i^{\ell_i}\}$ we define $\mathtt{NBits}(S_i, R_i) = \sum_{j=1}^{\ell_i} \left(|r_i^j| + |s_i^j|\right)$ to denote the total number of bits transferred between cache and memory during round i. Then we say the cost of the execution trace is

$$\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) = \sum_{i=1}^{t} \left(c_r k_i + \mathtt{NBits}(S_i, R_i)\frac{c_b}{w}\right) .$$

Intuitively, the c_r term is the cost of each random oracle query we make to the random oracle H and k_i is the number of queries at round i. The c_b term results from the messages passed between cache and memory — here c_b denotes the cost of transferring w bits between cache and memory.

We now formally define the energy cost of computing a function based on its execution trace.

Definition 3.2. *Given constants c_b and c_r, the energy cost* ecost *of a function $f_{G,H}$ is defined by*

$$\mathsf{ecost}_{q,\epsilon}(f_{G,H}, m \cdot w) = \min_{\mathcal{A},x} \mathbb{E}\left[\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x))\right],$$

where the expected cost is taken over the selection of the random oracle H, and the minimum of the expected cost is taken over all valid inputs x and all algorithms $\mathcal{A}$ with cache size $m \cdot w$ bits making at most q queries that compute $f_{G,H}(x)$ correctly with probability at least ϵ.

3.2 Red-Blue Extension Pebbling

We are now ready to prove our main result in this section. Theorem 3.3 lower bounds the energy cost $\mathsf{ecost}_{q,\epsilon}(f_{G,H}, m \cdot w)$ of the function $f_{G,H}$ with cache size $m \cdot w$ using $\mathsf{rbpeb}^{\parallel}(G, 9m)$ the red-blue pebbling cost of the DAG G with $9m$ red pebbles.

Theorem 3.3. *For any DAG G with n nodes and any $\mathcal{A}_{mw}^{H(.)}$ making at most $q < 2^{w/20}$ queries that compute $f_{G,H}(x)$ correctly with probability at least $\epsilon > 0$, if $20 \log n < w$ then,*

$$\mathsf{ecost}_{q,\epsilon}\left(f_{G,H}, m \cdot w\right) \geq \left(\frac{\epsilon}{16} - 2^{-2mw/5} - \frac{q+1}{2^w}\right) \mathsf{rbpeb}^{\parallel}(G, 9m).$$

Given a DAG G and a legal (partial) black pebbling $P = (P_1, \ldots, P_t)$ with $|P_{i+1} \setminus P_i| \leq m$ we say that a (partial) red-blue pebbling $((B_1, R_1), \ldots, (B_t, R_t))$ is a (m, k)-extension of P if for all $i \in [t]$ we have $|R_i| \leq m$ and we can find a small set $D_i \subseteq V(G)$ such that $|D_i| \leq k$ and $R_i \cup B_i = P_i \cup D_i$. We let $\mathbf{RBExt}(P, m, k)$ denote the set of all possible (m, k)-extensions of P. Observed that if $P \in \mathcal{P}^{\parallel}(G)$ is a complete black pebbling of G then $\mathbf{RBExt}(P, m, k) \subseteq \mathcal{RB}^{\parallel}(G, m)$ as any (m, k)-extension of P will be complete. To prove Theorem 3.3 we extract a legal partial black pebbling $P = (P_1, \ldots, P_t)$ from the execution trace of $\mathcal{A}^{H(\cdot)}$, and then use P to build a legal $(9m, 8m)$-extension pebbling $((B_1, R_1), \ldots, (B_t, R_t)) \in \mathbf{RBExt}(P, 9m, 8m)$ which may use up to $9m = (m + 8m)$ red-pebbles.

We then show how to *upper bound* the cost of the extension pebbling and *lower bound* the energy cost of the attacker $\mathcal{A}$ in the random oracle model.

Step 1: We start by using $\mathcal{A}_{mw}^{H(.)}$ to extract a legal (partial) black pebbling following Alwen and Serbinenko [8]. Given an execution trace $\mathsf{Trace}_{\mathcal{A},R,H}(x)$ we say that node $v \in V$ is an output at time $i+1$ if $\mathtt{prelab}_{H,x}(v) \in Q_i$ i.e., if v has parents $v_1, \ldots, v_d$ and the random oracle query $(v, \mathsf{lab}_{H,x}(v_1), \ldots, \mathsf{lab}_{H,x}(v_d))$ is submitted at the end of round i. Similarly, if $\mathtt{prelab}_{H,x}(v) \in Q_i$ where node v has parents $v_1, \ldots, v_d$ then we say that nodes $v_1, \ldots, v_d$ are inputs at time i i.e., the values $\mathsf{lab}_{H,x}(v_1), \ldots, \mathsf{lab}_{H,x}(v_d)$ can all be extracted from the random oracle query $\mathtt{prelab}_{H,x}(v) \in Q_i$ submitted at the end of round i. For a non-sink node v let $\mathtt{next}(i,v) = 1$ if v appears as an input at time i or if for some round $j > i$ node v appears as an input at time j and for all intermediate rounds $i < j' \leq j$ node v does not appear as an output during round j'; otherwise we set $\mathtt{next}(i,v) = 0$ i.e., if v never appears as an input in any future round $j \geq i$ or if the next time node v appears it appears as an output before node v will appear as an input. If v is a sink node then we will set $\mathtt{next}(i,v) = 1$ if and only if v is an output at time i. Now, given an execution trace $\mathsf{Trace}_{\mathcal{A},R,H}(x)$, the corresponding black pebbling $\mathtt{BlackPebble}^H\left(\mathsf{Trace}_{\mathcal{A},R,H}(x)\right) = P_0, \ldots, P_t$ is defined by setting $P_0 = \emptyset$ and $P_i = \{v \ : \ \mathtt{next}(i,v) = 1\}$ for each round $1 \leq i \leq t$. Intuitively, at each time j, P_j contains all nodes v whose label will appear as input to a future random oracle query *before* the label appears as the output of a random oracle query. We first observe that $|P_{i+1} \setminus P_i| \leq |Q_i|$ because if $v \in P_{i+1} \setminus P_i$ then v must have appeared as an output during round $i+1$ since $\mathtt{next}(i,v) = 0$ but $\mathtt{next}(i+1,v) = 1$. As we previously observed we only allow the attacker to make *at most* $|Q_i| \leq m$ queries during a single step because the attacker algorithm $\mathcal{A}_{mw}^{H(.)}$ only has cache-size $m \cdot w$ and must have room to store all of the responses in cache. Thus, $|P_{i+1} \setminus P_i| \leq m$ for all rounds $i < t$. Similarly, for all rounds i we have the total size of all queries $\sum_{j=1}^{k_i} |q_j^i|$ is *at most* mw because it must have enough room in cache to store the random oracle queries. Thus, $|\mathsf{parents}(P_{i+1} \setminus P_i)| \leq m$ for all rounds $i < t$.

Alwen and Serbineneko [8] showed that the black pebbling constructed this way is a legal partial black pebbling with probability at least $1 - q/2^w$ where q is the total number of random oracle queries. Intuitively, the only way for the extracted partial pebbling to not be legal is if a label appears out of order i.e., some node v appears as an input before it ever appears as an output. But this means that the random oracle query $\mathtt{prelab}_{H,x}(v)$ was never submitted. Thus, $\mathsf{lab}_{H,v}(x)$ can still be viewed as a uniformly random w-bit string and the probability of guessing it is at most 2^{-w}. The result then follows by a union bound over all q random oracle queries. We also observe that as long as for each node v the label $\mathsf{lab}_{H,x}(v)$ appears as a random oracle output at some point in time that the extracted pebbling will be complete. Note that if the extracted pebbling is legal, but incomplete then for some sink node v the query $\mathtt{prelab}_{H,x}(v)$ is never submitted and the attacker will guess the correct output with probability at most 2^{-w} since the output contains $\mathsf{lab}_{H,x}(v)$ which can still be viewed as a uniformly random w bit string. Thus, we will get a complete/legal pebbling with probability at least $\epsilon - q/2^w - 1/2^w$ where ϵ is the probability attacker computes function correctly.

Theorem 3.4. *[8] The pebbling extracted from an execution trace $(P_1, \ldots, P_t) = \mathtt{BlackPebble}^H\left(\mathsf{Trace}_{\mathcal{A},R,H}(x)\right)$ is a legal partial black pebbling with probability at least $1 - \frac{q}{2^w}$, where w is the label size and q is the number of queries made by $\mathsf{Trace}_{\mathcal{A},R,H}$. Furthermore, if for every node $v \in V$ the corresponding label $\mathsf{lab}_{H,x}(v)$ appears as an output of the random oracle H at some point in the execution trace then the pebbling is also complete i.e., $\mathtt{BlackPebble}^H\left(\mathsf{Trace}_{\mathcal{A},R,H}(x)\right) \in \mathcal{P}^{\parallel}(G)$. If $\mathcal{A}$ makes at most $|q_i| \leq m$ random oracle queries in each round of the execution trace then in each pebbling round $|P_{i+1} \setminus P_i| \leq m$.*

Formally, given P and an interval $[t_1, t_2]$ we let

$$\mathbf{QueryFirst}(t_1, t_2) = \bigcup_{i=t_1}^{t_2} \left(\mathsf{parents}\left(P_{i+1} \setminus P_i\right) \setminus \left(\bigcup_{j=t_1+1}^{i} \left(P_j \setminus P_{j-1}\right) \right) \right) .$$

Intuitively, we can think of $\mathsf{parents}(P_{i+1} \setminus P_i)$ as the set of inputs at time i and $P_j \setminus P_{j-1}$ as the outputs at time j so that $\mathbf{QueryFirst}(t_1, t_2)$ denotes the vertices v whose data-labels will appear as an input during rounds $[t_1, t_2]$ before the data-label appears as an *output* during the interval $(t_1, t_2]$. We will later see how we can extract the labels $\mathsf{lab}_{H,x}(v)$ for each node $v \in \mathbf{QueryFirst}(t_1, t_2)$ by simulating the attacker $\mathcal{A}^{H(.)}$

starting from round t_i. As an edge case notice that if a node v appears as an input at time t_1 and also as an output at time t_1 that v will still be in the set $\mathbf{QueryFirst}(t_1, t_2)$ — this is intended as our extractor begins simulation *after* v has appeared as an output and we will still be able to extract $\mathsf{lab}_{H,x}(v)$.

We present a few properties about **QueryFirst** that we will use in the rest of the proof.

Lemma 3.5. *Assume that $P = (P_1, \ldots, P_t)$ is a legal partial black pebbling of G then $\forall 0 \le x < y < z \le t$,*

$$\mathbf{QueryFirst}(y, z) \setminus \mathbf{QueryFirst}(x, z) \subseteq \bigcup_{i=x+1}^{y} (P_i \setminus P_{i-1}).$$

Proof. Consider a node $v \in \mathbf{QueryFirst}(y, z) \setminus \mathbf{QueryFirst}(x, z)$. Since $v \in \mathbf{QueryFirst}(y, z)$ there exists some round $i \in [y, z]$ such that $v \in \left(\mathsf{parents}\,(P_{i+1} \setminus P_i) \setminus \left(\bigcup_{j=y+1}^{i} (P_j \setminus P_{j-1})\right)\right)$. However, since $v \notin \mathbf{QueryFirst}(x, z)$ for any $i \in [x, z]$ we also have $v \notin \left(\mathsf{parents}\,(P_{i+1} \setminus P_i) \setminus \left(\bigcup_{j=x+1}^{i} (P_j \setminus P_{j-1})\right)\right)$. Therefore, $v \in \bigcup_{j=x+1}^{y} (P_j \setminus P_{j-1})$. $\square$

Step 2: We partition the pebbling rounds $[t]$ into sub time-intervals $(t_0 = 0, t_1], (t_1, t_2], \ldots$ recursively as follows. Let t_1 be the minimum pebbling round such that there exists $j < t_1$ such that $|\mathbf{QueryFirst}(j, t_1)| \ge 3m$. As a special case, if $|\mathbf{QueryFirst}(i, j)| < 3m$ for all $i < j \le t$ (i.e., no such intervals exist), then set $t_1 = t$ and output $(t_0, t_1]$. In this case, there is a red-blue extension pebbling in $\mathbf{RBExt}(P, 9m, 8m)$ that requires 0 blue moves and at most $\sum_i |P_i \setminus P_{i-1}|$ red-moves.

Once $t_1 < \ldots < t_{i-1}$ have been defined we inductively define $t_i > t_{i-1}$ to be the minimum round such that there exists $t_{i-1} < j \le t_i$ such that $|\mathbf{QueryFirst}(j, t_i)| \ge 3m$ — if no such t_i exists then we set $t_i = t$.

Step 3: We will show that there is an extension pebbling that makes at most $4m$ blue moves during each interval (except for the first one where it needs 0 blue moves). In particular, we set $k = 8m$ and we will define an extension pebbling $(B^*, R^*) \in \mathbf{RBExt}(P, 9m, k)$ by dividing the cache into two sets of size $4m$ and one size of m denoted as R_i^{inter}, R_i^{legal} and R_i^{new}, respectively. We will set $R_i = R_i^{\mathsf{legal}} \cup R_i^{\mathsf{inter}} \cup R_i^{\mathsf{new}}$, and show that $R_i \cup B_i \supset P_i$ gives a legal red-blue pebbling and then bound its cost.

We set $R_{t_i+1}^{\mathsf{inter}} = \{\}$ at the start of each time interval $(t_i, t_{i+1}]$ and for each $j \in (t_i + 1, t_{i+1}]$ we have

$$R_j^{\mathsf{inter}} = \left(R_{j-1}^{\mathsf{inter}} \cup (P_j \setminus P_{j-1})\right) \cap \mathbf{QueryFirst}(j, t_{i+1}).$$

Intuitively, R_j^{inter} stores all of the red-pebbles we have computed during the interval $(t_i + 1, j]$ that are later needed in the interval $[j, t_{i+1}]$. Thus, any node that is pebbled during rounds $(t_i + 1, j]$ and subsequently needed in round $[j, t_{i+1}]$ must be in R_j^{inter}, which we will keep in cache. Note that R_j^{inter} does not include the nodes that are computed at the start time $t_i + 1$ and we set $R_{t_i+1}^{\mathsf{inter}} = \{\}$. This is because the nodes we compute at time $t_i + 1$ that are later needed in $[t_i + 1, t_{i+1}]$ are in $\mathbf{QueryFirst}(t_i + 1, t_{i+1})$, and such nodes are stored in $R_j^{\mathsf{legal}} = \mathbf{QueryFirst}(t_i + 1, t_{i+1})$ for $j \in (t_i, t_{i+1}]$ as we will define below. This yields the following invariant.

Invariant 1. *For any $j \in (t_i, t_{i+1})$,*

$$\mathbf{QueryFirst}(j + 1, t_{i+1}) \cap \bigcup_{i=t_i+2}^{j} (P_i \setminus P_{i-1}) \subseteq R_j^{\mathsf{inter}}$$

To maintain legality across all time steps, we add a few rules about red and blue moves:

(1) We convert a pebbled node v from blue to red if node v is in $\mathbf{QueryFirst}(t_i + 1, t_{i+1})$. That is for any $j \in (t_i, t_{i+1}]$, we define $R_j^{\mathsf{legal}} = \mathbf{QueryFirst}(t_i + 1, t_{i+1})$.

(2) We set $R_j^{\mathsf{new}} = (P_j \setminus P_{j-1}) \setminus (R_j^{\mathsf{inter}} \cup R_j^{\mathsf{legal}})$ to be the nodes that are newly output at time j but not already in cache. This ensures that all nodes that are output at time j are pebbled even if this node won't be used as an input during the current time interval. A sink node may never appear as an input in any round, but as long as the black pebbling is complete we can guarantee that our red-blue extension pebbling is also complete i.e., every sink node is pebbled eventually. Finally, observe that in our case we will have $\left|R_j^{\mathsf{new}}\right| \le |P_j \setminus P_{j-1}| \le m$.

(3) Given a node $v \in R_j \setminus B_j$ such that v is in $\mathbf{QueryFirst}(t_{i'}+1, t_{i'+1})$ for some later interval $(t_{i'}, t_{i'+1}]$ with $i' > i$ we use a blue move to ensure that $v \in B_{j+1}$. We never remove blue pebbles so v can be converted back to a red node when required for the future interval $(t_{i'}, t_{i'+1}]$. (Note that a node may have both a red pebble and a blue pebble at the same time.) In this case, for accounting purposes, it will be helpful to "charge" the cost c_b of this blue move to the future interval $(t_{i'}, t_{i'+1}]$. More formally, we can set

$$B_{j+1} = B_j \cup \{v \in R_j \ : \ \exists i' \text{ .s.t. } (j \le t_{i'} \wedge v \in \mathbf{QueryFirst}(t_{i'}+1, t_{i'+1}))\} \ .$$

We show the following bound on the size of $\mathbf{QueryFirst}(j, t_{i+1})$. We remark that as long as the extracted pebbling P is legal both of the conditions $|\mathsf{parents}(P_{j+1} \setminus P_j)| \le m$ and $|P_{j+1} \setminus P_j| \le m$ will be satisfied. Intuitively, we can have at most m nodes appear as an output in each round since we only have space for m labels in cache. Similarly, we can have at most m nodes appear as input during each round for the same reason.

Lemma 3.6. *Assume that $P = (P_1, \ldots, P_t)$ is a legal partial black pebbling of G and that $|P_{j+1} \setminus P_j| \le m$ and $|\mathsf{parents}(P_{j+1} \setminus P_j)| \le m$ for all round $j < t$ then $\forall j \in (t_i, t_{i+1}]$, $|\mathbf{QueryFirst}(j, t_{i+1})| \le 4m$.*

Proof. By the definition of $\mathbf{QueryFirst}$, $\mathbf{QueryFirst}(j, t_{i+1}) \subseteq \mathbf{QueryFirst}(j, t_{i+1}-1) \cup \mathsf{parents}\left(P_{t_{i+1}+1} \setminus P_{t_{i+1}}\right)$ for $j \in (t_i, t_{i+1})$, and $\mathbf{QueryFirst}(j, t_{i+1}) \subseteq \mathsf{parents}\left(P_{t_{i+1}+1} \setminus P_{t_{i+1}}\right)$ for $j = t_{i+1}$. Due to our choice of t_{i+1}, $\mathbf{QueryFirst}(j, t_{i+1}-1) \le 3m$. Since $\mathsf{parents}\left(P_{t_{i+1}+1} \setminus P_{t_{i+1}}\right) \le m$ i.e., parallelism is bounded by cache size the lemma then follows. ☐

Lemma 3.7. $\left|R_j^{\mathsf{inter}}\right| \le 4m$.

Proof. Observe that $R_j^{\mathsf{inter}} \subseteq \mathbf{QueryFirst}(j, t_{i+1})$ since elements are only kept in R_j^{inter} if they are needed for some later pebbling round. $|\mathbf{QueryFirst}(j, t_{i+1})| \le 4m$ by Lemma 3.6. ☐

Also note that for any $j \in (t_i, t_{i+1}]$, $|R_j^{\mathsf{legal}}| = |\mathbf{QueryFirst}(t_i+1, t_{i+1})| \le 4m$ and $|R_j^{\mathsf{new}}| \subseteq |P_j \setminus P_{j-1}| \le m$. So the extension red-blue pebbling we constructed stores at most $9m$ labels in cache at any time.

Lemma 3.8. *Assume that $P = (P_1, \ldots, P_t)$ is a legal partial black pebbling of G and that $|P_{j+1} \setminus P_j| \le m$ for all round $j < t$ then the extension pebbling $(B^*, R^*) \in \mathbf{RBExt}(P, 9m, 8m)$ is a legal partial red-blue pebbling. Furthermore, if $P \in \mathcal{P}^{\parallel}(G)$ is a complete black pebbling then $(B^*, R^*) \in \mathcal{RB}^{\parallel}(G, 9m)$ is also complete.*

Proof. Let $R^* = (R_1, \ldots, R_t)$ where $R_j = R_j^{\mathsf{inter}} \cup R_j^{\mathsf{legal}} \cup R_j^{\mathsf{new}}$ and $B^* = (B_1, \ldots, B_t)$ be defined as above. For any time interval $(t_i, t_{i+1}]$ and any $j \in (t_i, t_{i+1}]$, first observe $\mathsf{parents}(P_{j+1} \setminus P_j) \subseteq \mathbf{QueryFirst}(j, t_{i+1})$. We now prove $\mathbf{QueryFirst}(j, t_{i+1}) \subseteq R_j$. Note that any node in $\mathbf{QueryFirst}(j, t_{i+1})$ must either be in $\mathbf{QueryFirst}(t_i+1, t_{i+1})$ or have been pebbled at some point during time steps $(t_i, j]$. In the former case, the node would be in R_j^{legal}, and in the latter case, the node would be in R_j^{inter}. Thus, $\mathsf{parents}(P_{j+1} \setminus P_j) \subseteq \mathbf{QueryFirst}(j, t_{i+1}) \subset R_j$.

Next we prove $R_{j+1} \setminus (R_j \cup B_j) \subseteq P_{j+1} \setminus P_j$. According to the definition of R_j^{inter}, R_j^{legal} and R_j^{new}, for $j \in (t_i, t_{i+1})$ during which $R_{j+1}^{\mathsf{legal}} = R_j^{\mathsf{legal}}$, we have $R_{j+1} \setminus (R_j \cup B_j) \subseteq (R_{j+1}^{\mathsf{inter}} \cup R_{j+1}^{\mathsf{new}} \cup R_{j+1}^{\mathsf{legal}}) \setminus (R_j^{\mathsf{inter}} \cup R_j^{\mathsf{new}} \cup R_j^{\mathsf{legal}}) \subseteq (R_{j+1}^{\mathsf{inter}} \cup R_{j+1}^{\mathsf{new}}) \setminus (R_j^{\mathsf{inter}} \cup R_j^{\mathsf{new}}) \subseteq P_{j+1} \setminus P_j$. For $j = t_{i+1}$ at which $R_{t_{i+1}+1}^{\mathsf{inter}} = \{\}$, note $R_{t_{i+1}+1} = R_{t_{i+1}+1}^{\mathsf{legal}} \cup R_{t_{i+1}+1}^{\mathsf{new}} \subseteq \mathbf{QueryFirst}(t_{i+1}+1, t_{i+2}) \cup (P_{t_{i+1}+1} \setminus t_{i+1}) \subseteq P_{t_{i+1}+1}$ and $P_{t_{i+1}} \subseteq R_{t_{i+1}} \cup B_{t_{i+1}}$. Thus, $R_{t_{i+1}+1} \setminus (R_{t_{i+1}} \cup B_{t_{i+1}}) \subseteq P_{t_{i+1}+1} \setminus P_{t_{i+1}}$.

Therefore, $\mathsf{parents}\left(R_{j+1} \setminus (R_j \cup B_j)\right) \subseteq \mathsf{parents}(P_{j+1} \setminus P_j) \subseteq \mathbf{QueryFirst}(j, t_{i+1}) \subset R_j$. Also, Invariant 1 guarantees that $B_{j+1} \setminus B_j \subseteq R_j$, i.e., any newly pebbled blue node at time $j+1$ is a red node at time j. Therefore, $\{R_j, B_j\}$ is a legal partial red-blue pebbling. Furthermore, if P is a complete black pebbling, then for any node $v \in V$ there exists a round j such that $v \in P_j \setminus P_{j-1}$. Recall that R_j^{new} is defined to be $(P_j \setminus P_{j-1}) \setminus (R_j^{\mathsf{inter}} \cup R_j^{\mathsf{legal}})$, indicating that $P_j \setminus P_{j-1} \subseteq R_j^{\mathsf{new}} \cup R_j^{\mathsf{inter}} \cup R_j^{\mathsf{legal}} = R_j$. Therefore, $V \subseteq \cup_{i=1}^{t} R_i$ and (B^*, R^*) is complete (i.e. legal red-blue pebbling).

□

We now bound the cost of the above extension pebbling. For any time $j \in (t_i, t_{i+1}]$, since we never discard necessary red pebbles from R_j^{inter} and R_j^{new} only contain unecessary nodes that are newly outputted at time j, the only cache-misses we incur come from $R_j^{\mathsf{legal}} = R_{t_i+1}^{\mathsf{legal}}$, at most $4m$. We "charge" double for every cache-miss to account for the previous blue move that initially placed a blue pebble on a node. This way, we can also charge the cost of placing new blue pebbles to future rounds. Therefore, the above extension pebbling has cost at most

$$8mc_b + \sum_{j \in (t_i, t_{i+1}]} c_r \left| P_j \setminus P_{j-1} \right| .$$

Step 4: To complete the proof, we show that during each interval any algorithm $\mathcal{A}$ must pay red-blue cost *at least* $mc_b + \sum_{j \in (t_i, t_{i+1}]} c_r \left| P_j \setminus P_{j-1} \right|$. Roughly speaking, we will set up an extractor that extracts $3m$ random oracle labels (i.e., $3mw$ truly random bits) by simulating $\mathcal{A}$ during this time interval. The extractor needs a hint of size $mw + w(\#words_i)$ bits where $\#words_i$ is the total amount of data (words) $\mathcal{A}$ transfers to/from cache. If $\#words_i \leq m$ then we will arrive at a contradiction as we compressed a random string of length $3mw$ — contradicting Lemma 3.1. Thus, $\mathcal{A}$ must pay blue cost *at least* mc_b during each interval, and by construction of $P = \mathtt{BlackPebble}^H\left(\mathsf{Trace}_{\mathcal{A},R,H}(x)\right)$ the red cost is at least $\sum_{j \in (t_i, t_{i+1}]} c_r \left| P_j \setminus P_{j-1} \right|$. We detail this step in the next section.

3.3 Extractor

We now use a compression argument to relate the cost of an execution trace to the cost of the red-blue extension pebbling. That is, an extractor with access to the attacking strategy, the state of the cache, and a few select hints can successfully predict a large number of random bits, contradicting Lemma 3.1. The hints we give the extractor will dictate the location of the random bits, and ensure these bits remain "random" (that is, not queried by the extractor). Figure 1 illustrates this setup. In particular, the extractor will use a hint to simulate $\mathcal{A}^{H(\cdot)}$ but this hint *does not* include the current state of memory ξ_i. Instead, the hint will encode the messages that the attacker expects to receive from main memory which allows us to *simulate* the attacker without storing the entire (large) state ξ_i.

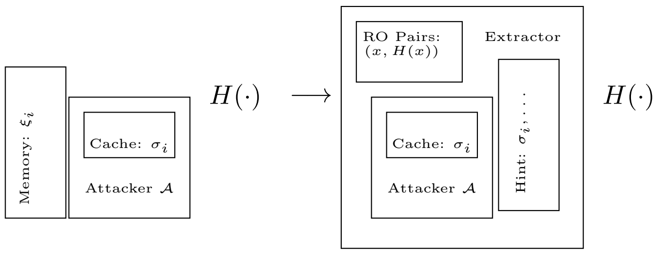

Fig. 1: Using the attacker to create an extractor that tries to predict $3m$ distinct outputs of random oracle $H(\cdot)$.

Let $t_0 = 0, t_1, \ldots, t_y = t$ denote the time intervals specified above. Intuitively, we expect that the evaluation algorithm needs to transfer at least mw bits to/from cache during each interval $(t_{i-1}, t_i]$ (potentially excluding the last interval $(t_{y-1}, t_y = t]$). Let $\mathtt{BadTrace}$ denote the event that we extracted a legal (partial) black pebbling $\texttt{BlackPebble}^H\left(\mathsf{Trace}_{\mathcal{A},R,H}(x)\right) = P_0, \ldots, P_t$, but that for some $i < y$ we did not transfer mw bits to/from cache during the interval $(t_{i-1}, t_i]$ i.e., for some $i < y$ we have

$$\sum_{i'=t_{i-1}+1}^{t_i} \sum_{j=1}^{\ell_{i'}} \left(|r_{i'}^j| + |s_{i'}^j| \right) \leq mw \ .$$

The following lemma shows that the event $\texttt{BadTrace}$ occurs with negligible probability so the attacker must transfer at least mw bits between cache and memory. Intuitively, if the event $\texttt{BadTrace}$ occurs then we can define an extractor which extracts at least $3m$ random oracle outputs using a hint of length *at most* $13mw/5$. By Lemma 3.1 it immediately follows that $\Pr[\texttt{BadTrace}] \leq 2^{-(3-13/5)mw}$. Note that in the edge case where $y = 1$ the event $\texttt{BadTrace}$ automatically does not occur. In this edge case we have $\textbf{QueryFirst}(i,j) < 3m$ for all $i < j \leq t$ and we also have $|P_i| \leq |\textbf{QueryFirst}(i,t) \cup (P_i \setminus P_{i-1})| \leq 4m$ for each round $i \leq t$. Thus, we can define an extension pebbling with 0 blue moves by setting $R_i = P_i$ and $B_i = \{\}$ for each round i.

Lemma 3.9. *If $q < 2^{w/20}$ and $20 \log n < w$ then $\Pr[\texttt{BadTrace}] \leq 2^{-2mw/5}$ where q upper bounds the total number of random oracle queries made in the execution trace, n is the number of nodes in the underlying DAG, and the probability is taken over the random coins of $\mathcal{A}$ and the selection of the random oracle H.*

Proof. Suppose, by way of contradiction, that for interval $(t_i, t_{i+1}]$ with $i + 1 < y$, an attacker transfers less than mw bits between cache and memory. We first note that, by definition of t_i and t_{i+1}, we can find some index j between t_i and t_{i+1} such that $|\textbf{QueryFirst}(j, t_{i+1})| \geq 3m$ and by Lemma 3.6 $|\textbf{QueryFirst}(j, t_{i+1})| \leq 4m$. We define an extractor that can predict $3m$ labels given access to the attacker's algorithm, the random oracle, and a small set of hints to help the extractor. Recall that for a non-sink node v with parents $v_1, \ldots, v_d$ we have

$$\mathsf{lab}_{H,x}(v) = H\left(\texttt{prelab}_{H,x}(v)\right) \quad \text{where} \quad \texttt{prelab}_{H,x}(v) = (v, \mathsf{lab}_{H,x}(v_1), \ldots, \mathsf{lab}_{H,x}(v_d)) \ .$$

Thus for nodes $y \neq z$, the prelabels $\texttt{prelab}_{H,x}(y) \neq \texttt{prelab}_{H,x}(z)$ are different. Thus, the values of $\mathsf{lab}_{H,x}(y)$ and $\mathsf{lab}_{H,x}(z)$ correspond to different inputs to H. That is, there are no *input* collisions and so the adversary must separately determine the hash outputs for each of the $3m$ inputs, which correspond to $3mw$ truly random bits in total.

The hint given to help the extractor consists of five components:

(1) The set $\textbf{QueryFirst}(j, t_{i+1})$ is given as a hint to denote the indices that form the string that the extractor will ultimately predict. Since $|\textbf{QueryFirst}(j, t_{i+1})| \leq 4m$, this component of the hint is at most $4m \log n$ bits.

(2) For each $v \in \textbf{QueryFirst}(j, t_{i+1})$, the index of the first query that appears in which $\mathsf{lab}(v)$ is needed as input. This component of the hint tells the extractor the queries that require the prediction of random strings, and has size at most $4m \log q$ bits, where $q = \sum_{i \leq t} k_i$ is the total number of queries made by the attacker.

(3) For each $v \in \textbf{QueryFirst}(j, t_{i+1})$, the index of the first query when $\mathsf{lab}(v)$ might be compromised. Observe that if the extractor successfully predicts a random string $\mathsf{lab}(v) = H(\texttt{prelab}(v))$ at a location $\texttt{prelab}(v)$, but then the query $\texttt{prelab}(v)$ is later queried by the attacker, the extractor will need to avoid submitting the query $\texttt{prelab}(v)$ if we still want to claim credit for predicting the string $\mathsf{lab}(v)$! To avoid this, we give the extractor a hint of the queries that would compromise the randomness of the desired locations i.e., (y, z) for the next query with $q_z^y = \mathsf{lab}(v)$. Since there are at most q queries we can encode each pair (i, y) using at most $\log q$ bits, and there at most $4m$ such pairs. Thus, this component of the hint tells the extractor the locations of the random strings to be predicted, and has size at most $4m \log q$ bits.

(4) The cache state σ_{j-1} given to $\mathcal{A}^{H(\cdot)}$ at time j is given as a hint to the extractor along with the answers A_{j-1} to the random oracle queries Q_{j-1} asked at the end of round $j - 1$. This allows the extractor

to simulate the attacker beginning at time step j. Since the cache has size m, each containing w-bit words, and A_{j-1} is additionally stored in cache the size of this component of the hint is at most $|\sigma_{j-1}|+k_{j-1}w \le mw$ bits where $k_{j-1}=|Q_{j-1}|$ denotes the number of random oracle queries asked at the end of round $j-1$.

(5) Messages between the cache and memory during time steps $[j, t_{i+1}]$ are also given as a hint to the extractor to simulate the attacker beginning at time step j. By assumption, the attacker transfers less than mw bits between cache and memory, so the size of this component of the hint is at most mw bits in total.

Since $q < 2^{w/20}$ and $20\log n < w$, then the total size, in bits, of the hint is at most

$$4m\log n + 4m\log q + 4m\log q + mw + mw \le \frac{13}{5}mw.$$

However, $|\mathbf{QueryFirst}(j, t_{i+1})| \ge 3m$, so the extractor successfully predicts the output of $3m$ hash outputs, each of size w, given a hint of size at most $\frac{13}{5}mw$ bits. By Lemma 3.1, such an extractor can succeed with probability at most $2^{-2mw/5}$ and, it immediately follows that $\Pr[\mathtt{BadTrace}] \le 2^{-2mw/5}$. □

We now justify the correctness of Theorem 3.3.

Proof of Theorem 3.3:
Consider an pROM algorithm $\mathcal{A}$ which computes $f_{G,H}(x)$ correctly with probability at least ϵ using at most mw bits of cache and making at most q random oracle queries. Let

$$\mathsf{Trace}_{\mathcal{A},R,H}(x) = \{(\sigma_i, \xi_i, R_i, S_i, Q_i)\}_{i=1}^{t}$$

be a randomly sampled execution trace, let $P = \mathtt{BlackPebble}^H(\mathsf{Trace}_{\mathcal{A},R,H}(x))$ be the corresponding ex-post facto (partial) black pebbling and let (B^*, R^*) be the corresponding red-blue $(9m, 8m)$-extension of P. We first note that in the special case that $|\mathbf{QueryFirst}(i,j)| \le 3m$ for all $i < j \le t$, we have $|\mathbf{QueryFirst}(i,t)| \le 3m$ and we also have $|P_i| \le |\mathbf{QueryFirst}(i,t) \cup (P_i \setminus P_{i-1})| \le 4m$ for all rounds $i \le t$. In this case we can simply set $R_i = P_i$ and $B_i = \{\}$ since the entire set fits in cache, and the red-blue pebbling (B^*, R^*) has 0 blue moves. In this special case it follows that

$$\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) \ge \sum_j c_r|Q_j| \ge \sum_j c_r|P_j \setminus P_{j-1}| \ge \mathsf{rbpeb}^{\parallel}((R^*, B^*)) \ .$$

Here, the second inequality follows from the observation that $|Q_j| \ge |P_j \setminus P_{j-1}|$ during each round j so the total red cost of the execution trace is at least $\sum_j c_r|Q_j| \ge \sum_j c_r\,|P_j \setminus P_{j-1}| \ge \sum_j c_r\,|R_j \setminus R_{j-1}|$. Otherwise, we can define the sequence $t_0 = 0, t_1, \ldots, t_y = t$ such that for all $1 \le i < y$ we can find $j \in (t_{i-1}, t_i]$ such that $|\mathbf{QueryFirst}(j, t_i)| \ge 3m$ and $y \ge 2$. Assuming the event $\mathtt{BadTrace}$ does not occur then for *all* $i < y$:

$$\sum_{i'=t_{i-1}+1}^{t_i} \sum_{j=1}^{\ell_{i'}} \left(|r_{i'}^j| + |s_{i'}^j|\right) \ge mw \ .$$

In particular, the execution trace transfers *at least* m (w-bit) blocks between cache and memory in between rounds t_{i-1} and t_i at cost c_b per w-bit block. Since this occurs for each $i < y$ the total cost incurred transferring data to/from cache is at least $(y-1)mc_b$. On the other hand the total number of blue moves in our pebbling is upper bounded by

$$2\left|\bigcup_{i=1}^{y} \mathbf{QueryFirst}(t_{i-1}+1, t_i)\right| \le 2\sum_{i=1}^{y} |\mathbf{QueryFirst}(t_{i-1}+1, t_i)| \le 8ym \ ,$$

since we make at most $|\mathbf{QueryFirst}(t_{i-1}+1, t_i)|$ blue moves at the beginning of each each time interval $(t_{i-1}+1, t_i]$ (converting blue pebbles to red pebbles) and never place a blue pebble on a node unless it is in **QueryFirst**$(t_{i-1}+1, t_i)$ for some future interval — once we place a blue pebble on a node it is never removed. Thus, in this case we have

$$\mathsf{rbpeb}^{\parallel}((R^*, B^*)) \leq 8ym \cdot c_b + \sum_{i=1}^{y} \sum_{j=t_{i-1}+1}^{t_i} c_r |Q_j| \leq 16(y-1)m \cdot c_b + 16 \sum_{i=1}^{y} \sum_{j=t_{i-1}+1}^{t_i} c_r |Q_j|$$

and

$$\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) \geq (y-1)m \cdot c_b + \sum_{i=1}^{y} \sum_{j=t_{i-1}+1}^{t_i} c_r |Q_j| \geq \mathsf{rbpeb}^{\parallel}((R^*, B^*))/16\ .$$

Note that P and (B^*, R^*) are both legal/complete with probability at least $\epsilon - q/2^w - 2^{-w}$. Thus, with probability at least $\epsilon - q/2^w - 2^{-w}$ we have $\mathsf{rbpeb}^{\parallel}((R^*, B^*)) \geq \mathsf{rbpeb}^{\parallel}(G, 9m)$. By Lemma 3.9, the event `BadTrace` occurs with with probability at most $2^{-2mw/5}$. It follows that, with probability at least $\epsilon - q/2^w - 2^{-w} - 2^{-2mw/5}$, that

$$\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) \geq \mathsf{rbpeb}^{\parallel}(G, 9m)/16\ .$$

Recall that $\mathsf{ecost}_{q,\epsilon}(f_{G,H}, m \cdot w) = \min_{\mathcal{A},x} \mathbb{E}[\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)]$ where the expectation is taken over the selection of the random oracle H and the minimum is taken over all algorithms that compute $f_{G,H}(x)$ correctly with probability at least ϵ. In particular, we have

$$\mathsf{ecost}_{q,\epsilon}(f_{G,H}, m \cdot w) \geq \left(\epsilon - q/2^w - 2^{-w} - 2^{-2mw/5}\right) \frac{\mathsf{rbpeb}^{\parallel}(G, 9m)}{16} \geq \left(\frac{\epsilon}{16} - 2^{-2mw/5} - \frac{q+1}{2^w}\right) \mathsf{rbpeb}^{\parallel}(G, 9m)\ .$$

□

4 Relating Memory Hardness and Bandwidth Hardness

In this section, we show that any function with high cumulative memory complexity also has high energy costs. Namely,

Reminder of Theorem 1.5.

$$\mathsf{rbpeb}^{\parallel}(G, m) \geq \min_t \left(2c_b \left(\frac{\Pi_{cc}(G)}{t} - 2m\right) + c_r t\right) \in \Omega\left(\sqrt{c_b \cdot c_r \cdot \Pi_{cc}(G)}\right),$$

where m is the cache size, t is the number of steps in the pebbling, c_b is the cost of a blue move and c_r is the cost of a red move.

We also show that this connection can be exploited to design a maximally bandwidth hard iMHF. Thus, the goals of designing an MHF with high cumulative memory complexity/bandwidth hardness are well aligned. As a warmup we show that (parallel) cumulative pebbling complexity $\Pi^{\parallel}_{cc}(G)$ can be used to lower bound bound the energy cost $\mathsf{rbpeb}^{\parallel}(G, m)$. By contrast, Theorem 1.5 uses the sequential black pebbling complexity $\Pi_{cc}(G)$ to lower bound $\mathsf{rbpeb}^{\parallel}(G, m)$. This is advantageous as we have $\Pi^{\parallel}_{cc}(G) \leq \Pi_{cc}(G)$ since for any graph $\mathcal{P}(G) \subseteq \mathcal{P}^{\parallel}(G)$. For some DAGs we have $\Pi^{\parallel}_{cc}(G) \ll \Pi_{cc}(G)$ e.g., there are constant indegree DAG G with n nodes for which $\Pi_{cc}(G) = \Omega(n^2)$ [31, 7, 14] while $\Pi^{\parallel}_{cc}(G) = o(n^2)$ for *any* any DAG G with constant indegree [2].

Lemma 4.1. $\mathsf{rbpeb}^{\parallel}(G, m) \geq \min_t \left(2c_b \left(\frac{\Pi^{\parallel}_{cc}(G)}{t} - m\right) + c_r t\right)$.

Proof. For any red-blue pebbling P of DAG G, let R_i be the set of red pebbles at time step i and let B_i be the set of blue pebbles at time step i. Setting $D_i = B_i \cup R_i$ we remark that $(D_1, \ldots, D_t)$ is a valid black pebbling of G. Thus, by the optimality of $\Pi^{\parallel}_{cc}(G)$,

$$\Pi^{\parallel}_{cc}(G) \le \sum_{i=1}^{t} |R_i \cup B_i| \le \sum_{i=1}^{t} |R_i| + \sum_{i=1}^{t} |B_i| \le t \max_i |B_i| + tm$$

Rearranging terms we have

$$\max_i |B_i| \ge \frac{\Pi^{\parallel}_{cc}(G)}{t} - m \ .$$

In the optimal red-blue pebbling, each blue pebble must eventually be converted back to a red pebble, or else it should be discarded. Additionally, without loss of generality, we can assume that during each step we make at least one red move. Otherwise, we could combine consecutive steps into one single step. Thus,

$$\mathsf{rbpeb}^{\parallel}(G,m) \ge 2\left|\cup_{i=1}^{t} B_i\right| c_b + tc_r \ge 2\max_i |B_i| c_b + tc_r \ge 2\left(\frac{\Pi^{\parallel}_{cc}(G)}{t} - m\right) c_b + tc_r$$
$$\ge \min_t \left(2\left(\frac{\Pi^{\parallel}_{cc}(G)}{t} - m\right) c_b + tc_r\right)$$

□

Corollary 4.2. *For an (e,d)-depth robust graph G,*

$$\mathsf{rbpeb}^{\parallel}(G,m) \ge \min_t \left(2\left(\frac{ed}{t} - m\right) c_b + tc_r\right).$$

Proof. An (e,d)-depth robust DAG G has $ed \le \Pi^{\parallel}_{cc}(G)$ [5]. □

We show that there exists a similar relationship between sequential black pebbling cost and sequential red-blue pebbling cost.

Theorem 4.3.
$$\mathsf{rbpeb}(G) \ge 2c_b\left(\frac{\Pi_{cc}(G)}{t} - m\right) + c_r t,$$
where m is the cache size, t is the number of steps in the pebbling, c_b is the cost of a blue move and c_r is the cost of a red move.

Proof. Let $(B_0,R_0),\ldots,(B_t,R_t)$ be an optimal sequential red-blue pebbling of our DAG G where R_i (resp. B_i) denotes be the set of red (resp. blue) pebbles at time step i. In the optimal red-blue pebbling, each blue pebble must eventually be converted back to a red pebble. Otherwise, we could reduce cost by discarding this pebble immediately contradicting our assumption of optimality. Thus, for an optimal red blue pebbling we have

$$\mathsf{rbpeb}(G) \ge 2\left|\bigcup_i B_i\right| c_b + tc_r \ge 2\max_i |B_i| c_b + tc_r \ .$$

Setting $P_i = B_i \cup R_i$ we remark that $(P_1,\ldots,P_t)$ is a valid sequential black pebbling of G. To see that the pebbling is sequential observe that since $B_i \subseteq R_{i-1} \cup B_{i-1}$ we have $P_i \setminus P_{i-1} \subseteq R_i \setminus R_{i-1}$ and therefore $|P_i \setminus P_{i-1}| \le |R_i \setminus R_{i-1}| \le 1$ i.e., we place at most one new pebble on the graph in each round. Then

$$\max_i |B_i| \ge \max_i (|P_i| - m) \ge \left(\frac{\Pi_{cc}(G)}{t} - m\right),$$

where the last step results from a simple averaging argument overall t steps. It immediately follows that

$$\mathsf{rbpeb}(G) \ge 2c_b\left(\frac{\Pi_{cc}(G)}{t} - m\right) + c_r t \ .$$

□

To prove Theorem 1.5 we establish a relationship between the cost of parallel and sequential red-blue pebblings in the following lemma:

Lemma 4.4. $\mathsf{rbpeb}(G, 2m) \leq \mathsf{rbpeb}^{\|}(G, m) \leq \mathsf{rbpeb}(G, m)$.

Proof. $\mathsf{rbpeb}^{\|}(G, m) \leq \mathsf{rbpeb}(G, m)$ follows immediately from definition.[10] Now consider $\mathsf{rbpeb}(G, 2m)$ and $\mathsf{rbpeb}^{\|}(G, m)$. Any parallel pebbling with cache size m can be performed by a sequential pebbling with cache size $2m$. Note that at any step, a parallel pebbling with cache size m can have at most m labels stored and m new pebbles placed in each step. Thus, a sequential pebbling with cache size $2m$ can emulate this by retaining the stored labels while adding the new pebbles one by one. □

Combining Theorem 4.3 and Lemma 4.4 yields Theorem 1.5.

Alwen and Blocki [2] show $\Pi^{\|}_{cc}(G) = \mathcal{O}\left(\frac{n^2 \log\log n}{\log n}\right)$ for any graph G with constant indegree. Moreover, there exists a family of DAGs $\{G_n\}_{n=1}^{\infty}$ with constant indegree with $\Pi_{cc}(G_n) \in \Omega(n^2)$ [31, 7].

We now show a relationship similar to Theorem 1.5 between the energy cost and cumulative memory cost [8] of an execution trace. Following [8]the cumulative memory cost of an execution trace is defined as:

$$\mathsf{cmc}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) = \sum |\alpha_i|,$$

where α_i encodes the state of the attacker [11] at round i. Similarly, following [8] we can define

$$\mathsf{cmc}_{q,\epsilon}(f_{G,H}) = \min_{\mathcal{A},R,x} \mathsf{cmc}(\mathsf{Trace}_{\mathcal{A},R,H}(x)),$$

where the minimum is taken over all $\mathcal{A}$ making at most q random oracle queries that compute $f_{G,H}$ correctly with probability at least ϵ.

We note that for $\mathsf{ecost}_{q,\epsilon}(f_{G,H})$ the minimum is taken over all $\mathcal{A}$ making at most q random oracle queries that compute $f_{G,H}$ correctly with probability at least ϵ *and* having cache size at most mw bits. Thus, any attacker that satisfies all of the restrictions for $\mathsf{ecost}_{q,\epsilon}(f_{G,H})$ will also satisfy our restrictions for $\mathsf{cmc}_{q,\epsilon}(f_{G,H})$ where there is no additional restriction on cache size. We emphasize that $\mathcal{A}$ can be an arbitrary pROM algorithm, so that the following result also applies to dMHFs such as `scrypt`.

Theorem 4.5. *For any execution trace* $\mathsf{Trace}_{\mathcal{A},R,H}(x)$ *of an algorithm* $\mathcal{A}$ *with cache size* mw *bits*

$$\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) \geq \left(\frac{\mathsf{cmc}(\mathsf{Trace}_{\mathcal{A},R,H}(x))}{tw} - m\right) c_b + tc_r,$$

where m *is the cache size,* t *is the number of steps,* c_b *is the cost of a blue move and* c_r *is the cost of a red move.*

Proof. Recall that the energy cost of an execution trace $\mathsf{Trace}_{\mathcal{A},R,H}(x) = \{(\sigma_i, \xi_i, R_i, S_i, Q_i)\}_{i=1}^{t}$ is defined as

$$\begin{aligned}\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) &= \sum_{i=1}^{t}\left(c_r|Q_i| + \frac{c_b}{w}\mathtt{NBits}(S_i, R_i)\right)\\ &\geq \max_i \frac{|\xi_i|}{w} c_b + tc_r \geq \left(\frac{\mathsf{cmc}(\mathsf{Trace}_{\mathcal{A},R,H}(x))}{tw} - m\right) c_b + tc_r\end{aligned}$$

[10]To see that $\mathsf{rbpeb}^{\|}(G, m)$ and $\mathsf{rbpeb}(G, m)$ are not identically equivalent quantities, consider the complete directed bipartite graph $K_{m,m}$ with m sources A and m sink nodes B(m is also the cache size). In the parallel model we can finish in two steps with zero blue moves: $R_0 = \emptyset$, $R_1 = A$, $R_2 = B$. In the sequential pebble game we would have to keep pebbles on A while we begin placing pebbles on B one by one. Each time we place a red-pebble on a node $y \in B$ we need to evict some node $x \in A$ by converting x into a blue node (and then bring it back into the cache-later).

[11]While there is no notion of a cache in the pROM model of [8], we could trivially set $\alpha_i = (\sigma_i, \xi_i)$ so that the state α_i explicitly includes the contents in cache σ_i as well as the content in main memory ξ_i.

The second step above follows from the observation that for all j we have $|\xi_j| \leq \sum_{i=1}^{j} \texttt{NBits}(S_i, R_i)$, and the third step follows from the observation that

$$\mathsf{cmc}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) - mtw = \sum_{i=1}^{t} (|\sigma_i| + |\xi_i|) - mtw \leq t \max_i |\xi_i| \,.$$

□

Let $z = \mathsf{cmc}(\mathsf{Trace}_{\mathcal{A},R,H}(x))$ and define $f(t) = \left(\frac{z}{tw} - m\right) c_b + tc_r$. We observe that the function f is minimized when we set $t = \sqrt{z\frac{c_b}{wc_r}}$ to balance out the terms tc_r and $\frac{z}{tw}c_b$. In particular, for any $t \geq 1$ we have $f(t) \geq 2\sqrt{\frac{\mathsf{cmc}(\mathsf{Trace}_{\mathcal{A},R,H}(x))\cdot c_r \cdot c_b}{w}} - mc_b$. It follows that for any trace $\mathsf{Trace}_{\mathcal{A},R,H}(x)$ we have

$$\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) \in \Omega\left(\sqrt{\frac{\mathsf{cmc}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) \cdot c_b \cdot c_r}{w}} - mc_b\right)$$

Alwen *et al.* [6] show that $\mathsf{cmc}_{q,\epsilon}(\texttt{scrypt}) \in \Omega(\epsilon n^2 \cdot w)$ for any $q > 0$ and $\epsilon > 2^{-w/2} + 2^{-n/20+1}$ as long as $4n^4 q \leq 2^{w/2}$. More specifically, they show that for some constant $C > 0$, any input x and any attacker $\mathcal{A}$ making at most $q \leq 2^{w/2-2}n^{-4}$ queries and evaluating $\texttt{scrypt}(x)$ correctly with probability at least ϵ (over $\mathcal{A}$'s random coins and the selection of the random oracle) that $\mathsf{cmc}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) \geq Cn^2 w$ with probability at least $\epsilon - 2^{-w/2} - 2^{-n/20+1}$ (over $\mathcal{A}$'s random coins and the selection of the random oracle). It follows that

$$\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) \geq \sqrt{\frac{n^2 w \cdot c_b \cdot c_r}{w}} - mc_b \geq n\sqrt{c_b c_r} - mc_b$$

with probability at least $\epsilon - 2^{-w/2} - 2^{-n/20+1}$ (over $\mathcal{A}$'s random coins and the selection of the random oracle). We remark that the actual bound from Alwen *et al.* [6] is slightly tighter, but also more complicated to state. We opted to use the above bounds to simplify the presentation.

Corollary 4.6. *There exists a constant $C > 0$ such that for any $m \leq n$ and any $0 < q \leq 2^{w/2-2}n^{-4}$ and $\frac{\epsilon}{2} > 2^{-w/2} + 2^{-n/20+1}$,*

$$\mathsf{ecost}_{q,\epsilon}(\texttt{scrypt}, m \cdot w) \geq C \cdot n\sqrt{c_b \cdot c_r} - mc_b.$$

While this lower bound for $\texttt{scrypt}$ is not tight, it is interesting in that it follows in a black box matter and highlights the connection between cumulative memory complexity and bandwidth hardness. We prove a tighter unconditional lower bound for $\texttt{scrypt}$ in Section 6, showing that $\mathsf{ecost}_{q,\epsilon}(\texttt{scrypt}) \in \Omega\left(n \cdot c_b\right)$. The proof of the tighter lower bound is substantially more involved.

5 Bandwidth Hardness of Candidate iMHFs

In this section, we provide lower bounds on the bandwidth hardness on the family of graphs generated by Argon2i [11], aATSample, and DRSample [4]. Given a DAG $G = ([n], E)$, a target set $T \subset [n]$ and red/blue subsets $B, R \subseteq [n]$ with $|R| \leq m$ we let $\mathsf{rbpeb}^{\parallel}(G, m, T, B, R)$ denote the red-blue cost to place red pebbles on a target set T starting from an initial red-blue pebbling configuration B, R.

5.1 Analysis Framework

We follow a similar strategy for each candidate construction by defining a target set $T_i = ((i-1)c\ell, ic\ell]$, and analyzing the structure of the DAG to lower bound the following quantity for that DAG:

$$\min_{R,B' \subseteq [(i-1)c\ell]:|R| \leq m} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T_i)|\, c_r\right)$$

We show in Theorem 5.2 that this quantity suffices to lower bound the bandwidth hardness. Intuitively, we can think of B (resp. R) as the initial set of blue (resp. red) pebbles on the graph when we start to pebble the target interval T_i and $B' \subseteq B$ as the set of blue pebbles that will be converted to red pebbles to help pebble the target interval T_i. Recall that $G - R - B'$ denotes the subgraph of G obtained by deleting all nodes in $R \cup B'$. If a node v is in $\mathsf{ancestors}_{G-R-B'}(T_i)$ then this node will need to be repebbled with a red pebble (at cost c_r) before we can finish pebbling T_i.[12] We will use Lemma 5.1 to help prove Theorem 5.2.

Lemma 5.1. *$\forall T, B, R \subseteq [n]$ such that $|R| \le m$ we have*

$$\mathsf{rbpeb}^{\parallel}(G, m, T, B, R) \ge \min_{B' \subseteq B} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r\right),$$

where c_b is the cost of a blue move and c_r is the cost of a red move.

Proof. Let $P = (B_0, R_0), (B_1, R_1) \ldots, (B_t, R_t)$ denote a legal red-blue pebbling sequence with starting configuration $(B_0, R_0) = (B, R)$ such that every node $v \in T$ in our target set is pebbled with a red-node at some point in the sequence i.e., $T \subseteq \bigcup_{i=0}^{t} R_i$. Let $B' \subseteq B_0 = B$ denote the subset of initially blue nodes that are eventually converted to red-pebbles during our sequence i.e., $B' = B \cap \left(\bigcup_{i=1}^{t} \{v \ : \ \texttt{s.t.}\ v \in R_i \setminus R_{i-1} \wedge \mathsf{parents}(v) \not\subset R_{i-1}\}\right)$. By definition, the pebbling sequence uses at least $|B'|$ blue moves at cost $|B'| c_r$.

Observe that we must place a red-pebble on all of the nodes in $\mathsf{ancestors}_{G-R-B'}(T)$ at some point. Note that any node $u \in \mathsf{ancestors}_{G-R-B'}(T)$ is not in $R_0 = R$ by definition as thus does not initially contain a red pebble. Similarly, we never use a blue-move to place a red-pebble on any node $u \in \mathsf{ancestors}_{G-R-B'}(T)$ by definition of B'. It follows that all of the nodes $\mathsf{ancestors}_{G-R-B'}(T)$ must be pebbled with red-pebbles in topological order. Thus, we have at least $|\mathsf{ancestors}_{G-R-B'}(T)|\, c_r$ red-moves and the cost of our pebbling sequence is at least $|B'| c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r$. It follows that

$$\mathsf{rbpeb}^{\parallel}(G, m, T, B, R) \ge \min_{B' \subseteq B} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r\right) .$$

□

Theorem 5.2. *Let $G = ([n], E)$ be any DAG such that $(j, j+1) \in E$ for each $j < n$, let c be a positive integer and let $T_i = ((i-1)c\ell + 1, ic\ell]$,*

$$\mathsf{rbpeb}^{\parallel}(G, m) \ge \sum_{i=1}^{\left\lfloor \frac{n}{c\ell} \right\rfloor} \min_{R, B' \subseteq [(i-1)c\ell] : |R| \le m} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T_i)|\, c_r\right) .$$

To prove Theorem 5.2, consider an optimal red-blue pebbling and let t_i denote the first time we place a pebble on node $ic\ell$. For each i, we use Lemma 5.1 to lower bound the red-blue cost incurred between steps $t_{i-1} + 1$ and t_i. See Appendix B for more details.

As expected, if $m = n$ then we have red-blue cost at most $\mathsf{rbpeb}^{\parallel}(G, m) \le nc_r$ for *any* graph G. Thus, we require some upper bound on m to establish lower-bounds for red-blue pebbling cost.

5.2 Underlying DAGs

We now describe each of the underlying DAGs whose energy complexity we analyze.

The underlying graph for Argon2iB [12] has a directed path of length n nodes. Each node i has parents $i-1$ and $r(i) = \left\lceil i \left(1 - \frac{x^2}{N^2}\right) \right\rceil$, where $N \gg n$ (in the implementation of Argon2iB we have $N = 2^{32}$) and x is chosen uniformly at random from $[N]$. See Algorithm 3 in Appendix A for a more formal description.

[12] To see this consider any directed path in $G - R - B'$ ending at some node $v \in T_i$ in our target set. By definition none of the nodes in this directed path contain a red-pebble at the start. While it is possible that some of the intermediate nodes on the path initially contain blue pebbles these pebbles on $B \setminus B'$ will *not* be converted to red-pebbles in a blue move (otherwise they would be in the set B' and would have already been deleted). Thus, to place a red-pebble on any node u in our path (including nodes in $B \setminus B'$) the parents of node u must first have a red-pebble. By backward induction each of the nodes on our path will need to be pebbled with a red-pebble (in topological order) before we can place a red-pebble on node v.

While Argon2iA (v1.1) is an outdated version of the password hash function it is still worthwhile to study for several reasons. First, the uniform edge distribution is a natural one which has been adopted by other iMHF constructions [17]. Second, it is possible that this older version of Argon2i may have seen some adoption. Each node i in Argon2iA has two parents: $i-1$ and $r(i) = i\left(1-\frac{x}{N}\right)$, where $N = 2^{32}$ and x is chosen uniformly at random from $[N]$. Thus, the parents in Argon2iA are slightly less biased towards closer nodes than in Argon2iB. See Algorithm 4 in Appendix A for a more formal description.

DRSample is a family of graphs $\mathbb{G}_n$ with $\Pi^{\parallel}_{cc}(G) \in \Omega\left(\frac{n^2}{\log n}\right)$ with high probability for any $G \in \mathbb{G}_n$. Like Argon2i and Argon2iB, the underlying graph for DRSample has a directed path of length n nodes. Each node i has parents $i-1$ and $r(i)$, but the distribution for $r(i)$ differs greatly from Argon2i and Argon2iB. Roughly speaking, DRSample samples an index j uniformly at random from $[1, \log i]$, an index k uniformly at random from $[1, 2^j]$, and sets $r(i) = i - k$. See Algorithm 1 in Appendix A for a more formal description.

A close relative to DRSample, aATSample [4] is also a family of graphs $\mathbb{G}_n$ with $\Pi^{\parallel}_{cc}(G) \in \Omega\left(\frac{n^2}{\log n}\right)$ with high probability for any $G \in \mathbb{G}_n$. aATSample modifies DRSample by starting with a copy of DRSample on $n/2$ nodes and appending another directed path with $\frac{n}{2}$ nodes that strategically connects to the first half of the graph so that the resulting cumulative pebbling complexity is high. The construction is parameterized by a constant $c > 0$ which specifies how nodes from the second half of the graph connect to nodes in the first half of the graph. See Algorithm 2 in Appendix A for a more formal description.

5.3 Argon2i

Let G be a random Argon2iB (or Argon2iA) graph and denote the incoming edges for each node i as $(i-1, i)$ and $(r(i), i)$). A key property that we will use in our analysis of Argon2iB is that for any $j < i-1$ we have $\Pr[r(i) = j] \geq \frac{1}{3n}$ and the selection of $r(i)$ is independent for each node i[13]. Similarly, for Argon2iA we have $\Pr[r(i) = j] \geq \frac{1}{n}$. This will be sufficient to lower bound the red-blue pebbling cost of Argon2iA and Argon2iB.

Lemma 5.3. *Let G be a random Argon2iB (resp. Argon2iA) graph with n nodes then for any $1 \leq j < i-1 \leq n$ we have $\Pr[r(i) = j] \geq \frac{1}{3n}$ (resp. $\Pr[r(i) = j] \geq \frac{1}{n}$).*

The proof of Lemma 5.3 is implicit in [16]. For completeness we include the proof in the appendix Appendix B.3.

Lemma 5.4. *Let $m \leq Cn^{1-\epsilon}$ for some constants $C > 0$ and $0 < \epsilon < 1$. Let $i > \frac{n}{2}$ and let $T = [i, i+\ell-1]$ be an interval of length $\ell \geq 150Cn^{1-\epsilon}$. Then a graph G generated by Argon2iB or Argon2iA satisfies the following with high probability:*

$$\min_{R, B' \subseteq [i-1] : |R| \leq m} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r\right) \geq \min\left(Cn^{1-\epsilon} c_b, \frac{n}{24} c_r\right).$$

Proof. We first consider casework on the size of B'. If $|B'| \geq Cn^{1-\epsilon}$, then the claim trivially holds as we have $|B'| c_b \geq Cn^{1-\epsilon} \cdot c_b$. Otherwise, we have $|B'| < Cn^{1-\epsilon}$, in which case $|R \cup B'| \leq |R| + |B'| < m + Cn^{1-\epsilon} \leq 2Cn^{1-\epsilon}$ since $|R| \leq m$. We then lower bound $|\mathsf{ancestors}_{G-R-B'}(T)|\, c_r$.

Partition the nodes in G into $\frac{n}{k}$ intervals $I_1, I_2, \ldots$ where $I_j \doteq [(j-1)k+1, jk]$ of k consecutive nodes for a parameter $k = \frac{n^\epsilon}{12C}$. For each interval I_j we let $L_j \doteq [(j-1)k + \lceil k/2 \rceil + 1, jk]$ (resp. $F_j \doteq [(j-1)k+1, (j-1)k + \lceil k/2 \rceil]$ denote the last half (resp. first half) of this interval. Now for each $j \in T$ define the random variable $X_j = 1$ if for some $i' \leq \frac{n}{2k}$ we have $r(j) \in L_{i'}$ and for all prior nodes $i \leq j' < j$ in the interval T we have $r(j') \notin E_{i'}$; otherwise $X_j = 0$. Intuitively, $X_j = 1$ if the edge $r(j)$ is connected to (the second half of) a new interval. Let $B_k = \{i' \;:\; |I_{i'} \cap (B' \cup R)| \geq 1\}$ be the set of intervals that contain some node in $B' \cup R$ and let $X = \sum_{j \in T} X_j$. Observe that there are at least $X - |B_k| - m \geq X - 2Cn^{1-\epsilon}$ intervals $I_{i'}$ such that

[13]The Argon2iB edge distribution depends on a parameter $N > n$. We make the mild assumption that $N \geq 6n$ where n is the number of nodes in the graph. This assumption would hold in practical instantations of Argon2iB i.e., the Argon2i implementation sets $N = 2^{32}$ while we would expect the running time parameter to be at most $n \leq 2^{24}$.

(1) the interval $I_{i'}$ contains no node in $B' \cup R$ i.e., $I_{i'} \cap (B' \cup R) = \{\}$, and (2) there is an edge $(r(j), j)$ with $j \in T$ and $r(j) \in L_{i'}$. For each such interval $I_{i'}$ the entire interval $F_{i'}$ is contained in $\mathsf{ancestors}_{G-R-B'}(T)$ because the graph G contains all directed edges of the form $(i, i+1)$ for $i < n$.

Thus,

$$|\mathsf{ancestors}_{G-R-B'}(T)| \geq \left(X - 2Cn^{1-\epsilon}\right)\frac{k}{2} \ .$$

We now argue that $X \geq \min\{\frac{n}{4k}, \frac{\ell}{50}\}$ with high probability. To see this observe that if $X_1 + \ldots + X_{j-1} \leq \frac{n}{4k}$ then there at least $\frac{n}{4k}$ of the intervals $I_1, \ldots I_{\frac{n}{2k}}$ are still "uncovered" and for each uncovered interval $I_{i'}$ we have

$$\Pr[r(j) \in F_{i'}] \geq \frac{|F_{i'}|}{3n} \geq \frac{k}{6n}$$

for Argon2iB and for Argon2iA we have

$$\Pr[r(j) \in F_{i'}] \geq \frac{|F_{i'}|}{n} \geq \frac{k}{2n} \ .$$

Thus, for Argon2iA we have

$$\Pr\left[X_j = 1 \middle| X_i + \ldots + X_{j-1} \leq \frac{n}{4k}\right] \geq \frac{k}{2n} \times \frac{n}{4k} \geq \frac{1}{24}$$

and for Argon2iB we have

$$\Pr\left[X_j = 1 \middle| X_i + \ldots + X_{j-1} \leq \frac{n}{4k}\right] \geq \frac{k}{6n} \times \frac{n}{4k} = \frac{1}{24} \ .$$

Since $\frac{n}{4k} = 3Cn^{1-\epsilon} \leq \frac{\ell}{50}$ we have $\min\{\frac{n}{4k}, \frac{\ell}{50}\} = \frac{n}{4k}$.

Concentration bounds imply that, except with negligible probability, we have $\sum_{j \in T} X_j \geq 3Cn^{1-\epsilon}$. To formalize the concentration bounds we can define new random variables $Y_j = 1$ iff $X_j = 1$ or $X_1 + \ldots + X_{j-1} \geq \frac{n}{4k}$. Observe that $X \geq \frac{n}{4k}$ if and only if $Y = \sum_{j \in T} Y_i \geq \frac{n}{4k}$ so it suffices to upper bound $\Pr[Y \leq \frac{n}{4k}]$. We can apply concentration bounds to upper bound $\Pr[Y \leq \frac{n}{4k}]$ (e.g., see Generalized Hoeffding Inequality [6, Claim 7]) because $\Pr[Y_j = 1 \mid (Y_i, \ldots, Y_{j-1}) = (y_i, \ldots, y_{j-1})] \geq \frac{1}{24}$ for *all* prior outcomes $y_i, \ldots, y_{j-1} \in \{0,1\}$. It follows that (whp) $X - 2Cn^{1-\epsilon} \geq Cn^{1-\epsilon}$ and

$$|\mathsf{ancestors}_{G-R-B'}(T)| \geq Cn^{1-\epsilon}\frac{k}{2} = \frac{n}{24} \ .$$

□

Reminder of Theorem 1.4. *Let G be a random Argon2iB (resp. Argon2iA) graph. Then there exists constants $C, C' > 0$ so that for any $0 < \epsilon < 1$ and for all $m \leq C'n^{1-\epsilon}$, with high probability,*

$$\mathsf{rbpeb}^{\|}(G, m) \geq C \cdot \min(nc_b, n^{1+\epsilon}c_r).$$

Proof of Theorem 1.4: Set $\ell = 150C'n^{1-\epsilon}$ so that $\frac{n}{\ell} = \Omega(n^\epsilon)$. Applying Lemma 5.4 to each of the disjoint $\frac{n}{\ell}$ intervals in the second half of graph G, the theorem follows from Theorem 5.2. □

5.4 DRSample

For DRSample [4] we rely on Lemma 5.5 to establish our main lower bound on the red-blue pebbling cost.

Lemma 5.5. *Suppose $m = \mathcal{O}(n^\rho)$ for some constant $0 < \rho < 1$ and $i > \frac{n}{2}$. Let $T = [i, i + \ell - 1]$ be an interval of length $\ell \geq 16m/(1-\rho)$. Then a graph generated by DRSample satisfies the following with high probability:*

$$\min_{R \subseteq [i-1]:|R| \leq m} \min_{B' \subseteq [i-1]} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r\right) \geq \min\left(\frac{(1-\rho)\ell}{8}c_b, \left(\frac{(1-\rho)\ell}{16}\right)\sqrt{\frac{n}{64\ell}}c_r\right)$$

Using Lemma 5.5, whose proof appears in Appendix B.2, we have:
Reminder of Theorem 1.3. *Let G be a graph generated by* $\mathsf{DRSample}$ *or* $\mathsf{aATSample}$ *and* $0 < \rho < 1$. *Then there exists constants $C, C' > 0$ so that for all $m \le C'n^\rho$, with high probability,*

$$\mathsf{rbpeb}^{\|}(G, m) \ge C \cdot \min\left(n \cdot c_b, n^{3/2-\rho/2} \cdot c_r\right).$$

Proof of Theorem 1.3: Applying Lemma 5.5 to each of the disjoint $\frac{n}{2\ell}$ intervals in the second half of graph G and observing that $\ell = \mathcal{O}(n^\rho)$, it follows from Theorem 5.2 that the cost is lower bounded by the minimum of $\frac{(1-\rho)\ell c_b}{8} \times \frac{n}{2\ell} = \Omega(nc_b)$ and $\frac{(1-\rho)\ell c_r}{16}\sqrt{\frac{n}{64\ell}} \times \frac{n}{2\ell} = \Omega\left(n^{3/2-\rho/2}\right)$

$$\mathsf{rbpeb}^{\|}(G, m) \ge \min\left(\Omega(n)c_b, \Omega(n^{3/2-\rho/2})c_r\right).$$

□

We also give an alternate bound for $\mathsf{DRSample}$ when the cache has size $m = \mathcal{O}(n^\rho/\log n)$ for any $0 < \rho \le 1$ in Appendix B.2 — see Theorem B.2. On the positive side the alternate bound applies when m is larger, but the cost terms in the lower bound are slightly weaker i.e., $\mathsf{rbpeb}^{\|}(G, m) \ge \min\left(\Omega(n/\log n)c_b, \Omega(n^2/(m\log n))\right)$. We remark that we cannot hope to obtain meaningful lower bounds for $m = \omega(n/\log n)$. In particular, Blocki et al. [14] gave a sequential black pebbling strategy for $\mathsf{DRSample}$ which uses space at most $Cn/\log N$ and time at most n. Thus, if $m \ge Cn/\log N$ this pebbling corresponds to red-blue pebbling strategy that uses no blue pebbles and has cost nc_r.

5.5 aATSample

The first $n/2$ nodes in a $\mathsf{aATSample}$ DAG [4] form a copy of $\mathsf{DRSample}$. Thus, our lower bounds from Section 5.4 also apply to $\mathsf{aATSample}$. For $\mathsf{aATSample}$ we can prove an additional lower bound which applies even when $m = \mathcal{O}\left(\frac{n}{\log n}\right)$ by utilizing the structure of the last $n/2$ nodes. Specifically, we rely on Lemma 5.6 to establish our additional lower bound in Theorem 1.2.

Lemma 5.6. *Let $i > \frac{n}{2}$ and $T = [i, i+\ell-1]$ be an interval of length $\ell = \frac{n}{\log n}$. Then for any parameters $c \ge 1$ and $m \le \frac{n}{16c\log n}$ a graph generated by* $\mathsf{aATSample}(n, c)$ *satisfies the following property:*

$$\min_{R,B' \subseteq [i-1]:|R| \le m} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r\right) \ge \min\left(\frac{n}{16c\log n}c_b, \frac{n}{8}c_r\right)$$

We now use Lemma 5.6, whose proof appears in Appendix B.1.
Reminder of Theorem 1.2. *Let G be a graph generated by* $\mathsf{aATSample}$. *Then there exists constants $C, C' > 0$ so that for all $m \le \frac{Cn}{\log n}$,*

$$\mathsf{rbpeb}^{\|}(G, m) \ge C' \cdot \min(n \cdot c_b, (n\log n)c_r),$$

holds with high probability.

Proof of Theorem 1.2: Applying Lemma 5.6 to each of the disjoint $\log n$ intervals in the second half of graph G, the theorem follows from Theorem 5.2. □

6 Bandwidth Hardness of `scrypt`

In this section, we prove an unconditional tight lower bound on the bandwidth hardness of a data-dependent MHF called `scrypt` [28], by analyzing the energy cost of its core subroutine ROMix (see Definition 6.1) in the parallel random oracle model. Specifically, we prove Theorem 1.6.

Reminder of Theorem 1.6. *Whenever $4\log n < w$, $q \le 2^{w/20}$, $\frac{n}{4m} \cdot c_r > c_b$, and $\epsilon \ge 2(\exp\left(-\frac{n}{8}\right) + \frac{3}{2}n^3 2^{-w} + qn^2 2^{-w} + 2^{-mw/5})$ the following statement holds in the parallel random oracle model:*

$$\mathsf{ecost}_{q,\epsilon}(\mathtt{scrypt}_n, m \cdot w) \ge \frac{\epsilon}{2} \cdot \frac{nc_b}{16}$$

The ROMix construction is shown in Definition 6.1. We abuse notation slightly and refer to this function as $\mathtt{scrypt}^H(X)$.

Definition 6.1. *[6] For a hash function $H : \{0,1\}^w \to \{0,1\}^w$, input $x \in \{0,1\}^w$, and parameter $n \in \mathbb{N}$, $\mathtt{scrypt}^H$ computes values $X_0, X_1, ..., X_{n-1}, Y_0, Y_1, ..., Y_n$ and outputs Y_n as follows:*

- *$X_0 = x$. $X_i = H(X_{i-1})$ for $i = 1, ..., n-1$.*
- *$Y_0 = H(X_{n-1})$. $Y_i = H(Y_{i-1} \oplus X_{Y_{i-1} \bmod n})$ for $i = 1, ..., n$.*

To bound the expected energy cost of $\mathtt{scrypt}_n$, we study the energy cost of each single execution trace running by an adversary algorithm to compute $\mathtt{scrypt}_n$. Unlike previous sections, we consider a deterministic adversary algorithm $\mathcal{A}_R$ where the adversary algorithm's internal randomness R is fixed in $\mathcal{A}_R$ to simplify the proof. Given an input x and a random oracle H, we define the execution trace determined by $\mathcal{A}_R$, H, and x to be $\mathsf{Trace}_{\mathcal{A}_R,H}(x) = \mathsf{Trace}_{\mathcal{A},R,H}(x)$. This simplification is without loss of generality because we quantify over all random coins R and inputs x. In particular, for any algorithm $\mathcal{A}$, input x and any R such that $\mathcal{A}_R$ computes $\mathtt{scrypt}_n$ correctly with probability $\epsilon > 0$ (over the choice of random oracle H), we can argue that $\mathsf{cost}\left(\mathsf{Trace}_{\mathcal{A}_R,H}(x)\right) = \Omega\left(\min\{c_b n, n^2 c_r/m\}\right)$, except with probability $\epsilon - \mu(w)$ for some negligible function μ.

We first make a couple of basic observations about the energy cost of computing $\mathtt{scrypt}$. The natural sequential evaluation algorithm runs in time $2n$ and incurs at least $n(1 - m/n) = \Theta(n)$ cache misses in expectation. Thus, the total cost is $\mathcal{O}\left(nc_r + nc_b\right)$. Similarly, we can define an evaluation algorithm that avoids storing labels in RAM memory entirely (i.e., to avoid cache misses). Instead the algorithm stores $\mathcal{O}\left(m\right)$ labels $X_0, X_{n/m}, X_{2n/m}, \ldots, X_n$ in cache. To compute Y_i we must recalculate $X_{Y_{i-1}}$, which can be accomplished using $\Theta(n/m)$ sequential calls to the random oracle (red moves). The total cost of computing $\mathtt{scrypt}_n$ in this way (without cache) would be $\Theta\left((n^2/m)c_r\right)$ in expectation. Notice that as the ratio n/m increases the cost of the cache-free evaluation algorithm quickly exceeds the cost of the naïve evaluation algorithm.

In our analysis we will assume that $\frac{n}{4m} \cdot c_r > c_b$. Theorem 6.2, our main result in this section, shows that if $n/(4m) \ge c_b/c_r$ then any algorithm in the parallel random oracle model has energy cost at least $\Omega\left(nc_r + nc_b\right)$ i.e., $\mathtt{scrypt}$ is maximally bandwidth hard. If $n/m \ll c_b/c_r$ then an attacker would prefer to use the cache free evaluation algorithm and $\mathtt{scrypt}$ is not maximally bandwidth hard for these parameter settings. However, in practice we would expect that our condition $n/(4m) \ge c_b/c_r$ holds e.g., $c_b/c_r \approx 250$ [30], $n = 2^{20}, m = 2^{10}$. We make several other reasonable assumptions about the parameters n, w and q (#attacker random oracle queries) in our analysis i.e., we assume $4\log n < w$, $q \le 2^{w/20}$.

Theorem 6.2. *For any input $x \in \{0,1\}^w$ and $n \ge 2$, if $\frac{n}{4m} \cdot c_r > c_b$ and $\mathcal{A}_R^H(x, n)$ outputs $Y_n = \mathtt{scrypt}^H(x, n)$ correctly with probability at least ϵ, taken over the choice of the random oracle H, then with probability (over the choice of H) at least $\epsilon - \exp\left(-\frac{n}{8}\right) - \frac{3}{2}n^3 2^{-w} - qn^2 2^{-w} - 2^{-mw/5}$, we have*

$$\mathsf{cost}\left(\mathsf{Trace}_{\mathcal{A}_R,H}(x)\right) \ge \frac{nc_b}{16} .$$

Theorem 1.6 is a corollary that can be derived directly from Theorem 6.2, since $\mathsf{ecost}_{q,\epsilon}(\mathtt{scrypt}_n, m \cdot w) \ge \left(\epsilon - \exp\left(-\frac{n}{8}\right) - \frac{3}{2}n^3 2^{-w} - qn^2 2^{-w} - 2^{-mw/5}\right) \cdot \frac{nc_b}{16} \ge \frac{\epsilon}{2} \cdot \frac{nc_b}{16}$, where we assume $\epsilon \ge 2(\exp\left(-\frac{n}{8}\right) + \frac{3}{2}n^3 2^{-w} + qn^2 2^{-w} + 2^{-mw/5})$. Theorem 1.6 implies that any algorithm $\mathcal{A}$ that always computes $\mathtt{scrypt}^H(x, n)$ correctly has expected energy cost *at least* $\mathbb{E}\left[\mathsf{cost}(\mathsf{Trace}_{\mathcal{A}_R,H}(x))\right] \ge \frac{nc_b}{32}$, where $\mathcal{A}_R(x) := \mathcal{A}(x; R)$ and the expectation is taken over the selection of $\mathcal{A}$'s random coins R. Similarly, an algorithm that only computes the answer correctly half of the time has expected energy cost *at least* $\mathbb{E}\left[\mathsf{cost}(\mathsf{Trace}_{\mathcal{A}_R,H}(x))\right] \ge \frac{nc_b}{64}$.

We start the proof with considering the ways an attacker might hope to compute $\texttt{scrypt}^H(x, n)$. We expect that any algorithm that computes the output Y_n correctly must first compute the labels $X_1, \ldots, X_{n-1}, Y_0, Y_1, \ldots, Y_n$ in order i.e., if $j > i$ we expect that X_i (resp. Y_i) will appear as the output of a random oracle query before X_j (resp. Y_j) and we expect that *all* X_i's appear before *any* Y_j. However, if the attacker is lucky some of the labels might appear out of order and we will not be able to lower bound the attacker's cost e.g., if Y_j happens to be the output of some random oracle query before Y_{j-1} appears for the first time. We introduce two bad events "Collision" and "Wrong Order" to analyze (and upper bound) the probability that the attacker gets lucky.

Notation: We define $\mathcal{S} = |\mathcal{H}|$, where $\mathcal{H}$ is the set of all possible random oracles H. We will use a superscript H on a label to indicate that the label is generated by $\mathcal{A}^H$. We may omit the superscript H when it is clear which random oracle H is used to generate this label. Also for simplicity, we abuse notation slightly and refer to $\mathcal{A}_R$ as $\mathcal{A}$, and $\mathsf{Trace}_{\mathcal{A}_R,H}(x)$ as $\mathsf{Trace}_{\mathcal{A},H}(x)$.

Collision. For each $0 \leq i \leq n$, we define the set $\texttt{COLLISION}_i \subseteq \mathcal{H}$ such that a random oracle $H \in \texttt{COLLISION}_i$ if and only if there are collisions among the labels $X_0, X_1,^H ..., X_{n-1}^H, T_1^H, ..., T_i^H$ as input queries to H. (Denote $T_k^H = Y_{k-1}^H \oplus X^H_{Y^H_{k-1} \bmod n}$ for all $1 \leq k \leq i$ and $T_0^H = X_{n-1}^H$.) According to the definition, we have $\texttt{COLLISION}_0 \subseteq \texttt{COLLISION}_1 \subseteq \cdots \subseteq \texttt{COLLISION}_n$.

Wrong Order. For each $0 \leq i \leq n$, we define the set $\texttt{WRONGORDER}_i \subseteq \mathcal{H}$ such that a random oracle $H \in \texttt{WRONGORDER}_i$ if and only if there exists $k < i$ such that $T_{k+1}^H = Y_k^H \oplus X^H_{Y^H_k \bmod n}$ appears as an input query to H earlier than or in the same round of T_k^H. According to the definition, we have $\texttt{WRONGORDER}_0 \subseteq \texttt{WRONGORDER}_1 \subseteq \cdots \subseteq \texttt{WRONGORDER}_n$.

Alwen *et al.* [6] proved the following two results about the size of the sets $\texttt{COLLISION}_n$ and $\texttt{WRONGORDER}_n$, which will be useful for our analysis.

Lemma 6.3. *[6, Claim 15]* $|\texttt{COLLISION}_n| \leq \mathcal{S} \cdot \frac{3}{2} n^3 2^{-w}$.

Lemma 6.4. *[6, Claim 18]* $|\texttt{WRONGORDER}_n \setminus \texttt{COLLISION}_n| \leq \mathcal{S} \cdot q n^2 2^{-w}$.

To prove Theorem 6.2, we will show that the energy cost of an execution trace in which $\mathcal{A}$ correctly outputs $\texttt{scrypt}^H(x, n)$ is at least $\frac{nc_b}{4}$ with a high probability over the choice of the random oracle H. Before we further describe the proof, it will be helpful to introduce a special way to sample a random oracle H uniformly at random.

Sampling H. Intuitively, an easy way to construct a random oracle H is randomly choosing one from the set $\mathcal{H}$ of all random oracles. To prove our main result, it will be helpful to think of H as being sampled in a different (but equivalent) way as suggested by Alwen *et al.* [6]. In particular, Alwen *et al.* [6] iteratively define a sequence $H_0, H_1, \ldots$ of random oracles and proved that each individual H_i (when viewed alone) can be viewed as a uniformly random from $\mathcal{H}$. Specifically, we define $H_0, ..., H_n$ as follows:

(1) Choose oracle H_0 uniformly at random.

(2) Choose challenges $c_1, ..., c_n$ uniformly at random in $\{0, 1, ..., n-1\}$.

(3) Construct $H_1, ..., H_n$ in order. For $i < n$:

   (a) If $H_i \in \texttt{COLLISION}_i$, let $H_{i+1} = H_i$.

   (b) If $H_i \notin \texttt{COLLISION}_i$, let $H_{i+1} = H_i$ except that $H_{i+1}(T_i^{H_i}) = \lfloor \frac{Y_i^{H_i}}{n} \rfloor \times n + c_{i+1} = Y_i^{H_{i+1}}$, where $T_i^{H_i} = Y_{i-1}^{H_i} \oplus X_{Y^{H_i}_{i-1} \bmod n}$, and the superscript H_i shows the value is generated using random oracle H_i. For simplicity of presentation we will assume that n is a power of 2 so that we can avoid rounding issues.

Note that if $H_i \notin \texttt{COLLISION}_i \cup \texttt{WRONGORDER}_i$ for all $i \le n$, then $\mathsf{Trace}_{\mathcal{A},H_n}(x)$ is identical to $\mathsf{Trace}_{\mathcal{A},H_i}(x)$ until the time when T_i first appears as a query to the random oracle. Alwen *et al.* [6] gave a simple inductive proof of this claim. While our notion of an execution trace is slightly different (due to the presence of a cache) we remark that the exact same argument carries over. This observation will be useful later.

To evaluate the energy cost of an entire execution trace $\mathsf{Trace}_{\mathcal{A},H}(x)$, we divide it into n partial execution traces and lower bound the (expected) energy cost of each partial execution trace. See explanation below.

Partial Trace. The following notion of a partial trace will be useful in our security proof. Given a trace $\mathsf{Trace}_{\mathcal{A},H}(x) = \{(\sigma_j, \xi_j, R_j, S_j, Q_j)\}_{j=1}^{t}$ and a label index $i \neq n$, we use t_i to denote the first round in which $T_i = Y_{i-1}^{H_i} \oplus X_{Y_{i-1}^{H_i} \bmod n}$ appears as a query to random oracle H. Lemma 6.4 prove that only a few random oracles H cause that T_{i+1} is first queried before T_i (or that T_i is never queried in which case $t_i = \infty$), which allows us to define the partial trace $\mathsf{Trace}_{\mathcal{A},H,i}(x) = \{(\sigma_j, \xi_j, R_j, S_j, Q_j)\}_{j=t_i}^{t_{i+1}-1}$ as the execution trace between rounds t_i and t_{i+1} for $t_{i+1} > t_i$. When $t_i < t_{i+1} \neq \infty$ for each $i < n$ (which is true for $H \notin \texttt{WRONGORDER}_n$) we have

$$\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},H}(x)) \ge \sum_{i=0}^{n-1} \mathsf{cost}(\mathsf{Trace}_{\mathcal{A},H,i}(x)) \ .$$

Lucky Partial Trace. We say that the partial trace $\mathsf{Trace}_{\mathcal{A},H,i}(x)$ is "lucky" if $\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},H,i}(x)) < \frac{c_b}{4}$. We remark that $\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},H,i}(x)) \ge (t_{i+1} - t_i)c_r$ as there is *at least* one query to the random oracle in each round. Similarly, if $\sum_{j=t_i}^{t_{i+1}-1} \texttt{NBits}(S_j, R_j) \ge w/4$ bits are transferred between memory and cache during $\mathsf{Trace}_{\mathcal{A},H,i}(x)$ then we have $\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},R,H}(x)) \ge c_b/4$. Thus if $\mathsf{Trace}_{\mathcal{A},H,i}(x)$ is "lucky", then at most $w/4$ bits are transferred between memory and cache while $t_{i+1} - t_i \le \frac{c_b}{4c_r}$. Next, we will use concentration bounds along with an extractor argument to show that for almost all random oracles, at least $\frac{n}{4}$ of the partial traces in the entire trace are not lucky. Then total energy cost of such a trace is at least

$$\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},H}(x)) \ge \sum_{i:L_i=1} \mathsf{cost}(\mathsf{Trace}_{\mathcal{A},H,i}(x)) \ge \frac{n}{4} \cdot \frac{c_b}{4} = \frac{nc_b}{16} \ .$$

To analyze the energy cost of a partial trace $\mathsf{Trace}_{\mathcal{A},H,i}(x)$, we define $B_i \subseteq [n]$ be the set of indices k of the labels X_k that appear "out of thin air" during the following simulation:

(1) Give a random oracle $H_i \in \mathcal{H}$ which is chosen uniformly at random in advance. H_i is chosen using $H_0, c_1, ..., c_i$ as we describe in the paragraph of "Sampling H" above.

(2) Define n random oracles $H_{i+1,0}, H_{i+1,1}, \cdots, H_{i+1,n-1}$. For each $j < n$, let $H_{i+1,j} = H_i$ except that $H_{i+1,j}(T_i^{H_i}) = \lfloor \frac{Y_i^{H_i}}{n} \rfloor \times n + j = Y_i^{H_{i+1}}$. Intuitively, $H_{i+1,j}$ is "programmed" to ensure that X_j is required to compute the next label. Consider t_i to be the initial round of the partial trace using oracle H_i. For each $j < n$, the extractor simulates in parallel the process of running $\mathcal{A}$ with random oracle $H_{i+1,j}$ by running $\mathcal{A}$ with random oracle H_i and replacing the output of query $T_i^{H_i}$ (i.e. $Y_i^{H_i}$) with $\lfloor \frac{Y_i^{H_i}}{n} \rfloor + j$.

(3) Note that $H_{i+1,j}$ (for $j < n$) only differ from H_i at the query T_i. Since t_i is the first round that T_i is queried, the execution traces of $H_{i+1,j}$ (for all $j < n$) till the round t_i are the same, as well as the initial states at t_i.

(4) Stop simulating $\mathcal{A}^{H_{i+1,j}}$ when at least one of the cases below happens:

   (a) X_j is first queried.

   (b) $T_{i+1}^{H_{i+1,j}}$ is first queried. (In this case, X_j can be obtained by computing $X_j = T_{i+1}^{H_{i+1,j}} \oplus Y_i^{H_{i+1,j}}$.)

   (c) The algorithm transfers more than $\frac{w}{4}$ bits between cache and memory.

(5) Note that the total rounds in simulating $\mathcal{A}^{H_{i+1,j}}$ (for any $j < n$) is no larger than $t_{i+1} - t_i$. (t_i and t_{i+1} here means the t_i and t_{i+1} defined in the execution trace of $\mathcal{A}^{H_{i+1,j}}$.)

If (during the above) simulation, the label X_k appears as an input to a random oracle query before it appears as output, then $k \in B_i$. We can use an extractor argument to upper bound the size of $|B_i|$. In particular, our extractor will be given a hint of size $|B_i|\left(2\log n + \log q + 1 + \frac{w}{4}\right) + mw$ and for each node $v \in B_i$ our extractor will output the pair $(X_{v-1}, H(X_{v-1}) = X_v)$ *without* ever querying the random oracle at X_{v-1}. If $|B_i| > 8m$ for most of the traces, then we obtain a contradiction as any extractor should succeed with probability at most $2^{-|B_i|w + |B_i|(2\log n + \log q + 1 + \frac{w}{4}) + mw} \ll 1$.

To accomplish this task, we will give the extractor a hint that includes the initial state of $\mathcal{A}$ (necessary for simulation), the set B_i and for each $v \in B_i$ the challenge j_v s.t. X_v appears out of thin air during the execution of $\mathcal{A}^{H_{i+1,j_v}}$ (as well as the index of the relevant query where X_v appears out of thin air). The hint also includes an encoding of the messages passed between cache and memory during each relevant execution $\mathcal{A}^{H_{i+1,j_v}}$. In more detail, the hint given to help the extractor consists of the following components:

(1) The set B_i is given as a hint to denote the indices that form the string that the extractor will ultimately predict. This component of the hint is $|B_i|\log n$ bits.

(2) For each $v \in B_i$, the challenge j_v for which label X_v appears out of thin air in the execution trace of $\mathcal{A}^{H_{i,j_v}}$, i.e. $\mathsf{Trace}_{\mathcal{A},R,H_{i,j_v},i}(x)$. If there are multiple values of j_v for which X_v appears out of thin air we break ties by selecting the challenge j_v for which the label X_v appears out of thin air in the earliest round. This component of the hint is at most $|B_i|\log n$ bits.

(3) For each X_v, $v \in B_i$, the index of the first query z_v in which X_v appears out of thin air in the execution trace of $\mathcal{A}^{H_{i,j_v}}$, i.e. $\mathsf{Trace}_{\mathcal{A},R,H_{i,j_v},i}(x)$. This component of the hint allows the extractor to extract the random string X_v, and has size at most $|B_i|\log q$ bits, where q is the total number of queries made by the attacker.

(4) For each X_v, $v \in B_i$, when running $\mathcal{A}^{H_{i+1,j_v}}$ the extractor needs one bit indicating whether X_v first appears as a query by itself or as part of the query $T_{i+1}^{H_{i+1,j_v}} = X_v \oplus Y_i^{H_{i+1,j_v}}$ In the latter case the extractor needs to obtain X_v by computing $X_v = T_{i+1}^{H_{i+1,j_v}} \oplus Y_i^{H_{i+1,j_v}}$. The size of this component of the hint is at most $|B_i|$ in total.

(5) The cache state at t_i is given as a hint to the extractor to simulate the attacker beginning at time step t_i. Since the cache has size m, each containing w-bit words, the size of this component of the hint is at most mw bits.

(6) For each $v \in B_i$, the hint includes the messages passed between cache and memory during rounds $[t_i, t_{i+1})$ of execution trace of $\mathcal{A}^{H_{i,j_v}}$ where j_v was the index of the challenge for which X_v appears out of thin air. Since we have restricted our attention to execution traces in which the attacker transfers less than $\frac{w}{4}$ bits between cache and memory when computing a challenge, then the size of this component of the hint is at most $\frac{|B_i|w}{4}$ bits in total.

The total size, in bits, of the hint is at most

$$|B_i|\log n + |B_i|\log n + |B_i|\log q + |B_i| + mw + \frac{|B_i|w}{4} = |B_i|\left(2\log n + \log q + 1 + \frac{w}{4}\right) + mw$$

To know in which cases $|B_i| < 8m$, we first consider the cases of $|B_i| \geq 8m$. The first case is $H_i \in \mathtt{COLLISION}_i \cup \mathtt{WRONGORDER}_i$. The second case is that the extractor successfully predicts $8m$ labels. We define the set of random oracles in this case to be $\mathtt{PREDICTABLE}$. Then we have $|B_i| < 8m$ for all $H_i \notin \mathtt{COLLISION}_i \cup \mathtt{WRONGORDER}_i \cup \mathtt{PREDICTABLE}$. The set $\mathtt{PREDICTABLE}$ is formally defined below.

Predictable. We define a set $\texttt{PREDICTABLE}$ containing all random oracles H for which there exists a hint with length $|B_i|\left(2\log n + \log q + 1 + \frac{w}{4}\right) + mw$ such that $|B_i| \geq 8m$, i.e. the extractor can correctly output at least $8m$ labels among $X_1, ..., X_{n-1}$ using this hint without querying them.

Lemma 6.5. $|\texttt{PREDICTABLE}| \leq \mathcal{S} \cdot 2^{-mw/5}$.

Proof. Since the number of bits we want to predict is $8mw$, the size of the hint is $|B_i|(2\log n + \log q + 1 + \frac{w}{4}) + mw$, using Lemma 3.1 we can bound the size of $\texttt{PREDICTABLE}$:

$$|\texttt{PREDICTABLE}| \leq \mathcal{S} \cdot 2^{-|B_i|w + \left(|B_i|(2\log n + \log q + 1 + \frac{w}{4}) + mw\right)} \leq \mathcal{S} \cdot 2^{-8mw + \left(8m\cdot(2\cdot\frac{w}{4} + \frac{w}{20} + 1 + \frac{w}{4}) + mw\right)} \leq \mathcal{S} \cdot 2^{-mw/5}.$$

The second inequality holds under our assumption that $4\log n < w$, $q \leq 2^{w/20}$. □

Lemma 6.6 says that the probability that our partial execution trace is "unlucky" is at least $\frac{1}{2}$. This holds even if we condition on any $H_0, c_1, \ldots, c_i$ choice of prior challenges so long as H_i is not in our bad set of random oracles ($H_i \notin \texttt{COLLISION}_i \cup \texttt{WRONGORDER}_i \cup \texttt{PREDICTABLE}$) — these conditional probabilities allow us to apply concentration bounds in the next step of the proof.

Lemma 6.6. *For any $i < n$, any $H_0, c_1, ..., c_i$ s.t. $H_i \notin \texttt{COLLISION}_i \cup \texttt{WRONGORDER}_i \cup \texttt{PREDICTABLE}$,*

$$\Pr\left[t_{i+1} - t_i \geq \frac{n}{16m} \vee \sum_{j=t_i}^{t_{i+1}-1} \texttt{NBits}(S_j, R_j) \geq w/4 \,\middle|\, H_0, c_1, ..., c_i\right] \geq \frac{1}{2},$$

where the probability is taken over the choice of $c_{i+1} \in \{0, \ldots, n-1\}$ and $\sum_{j=t_i}^{t_{i+1}-1} \texttt{NBits}(S_j, R_j) > w/4$ means more than $w/4$ bits are transferred between cache and memory between rounds $[t_i, t_{i+1})$.

Proof. Since H_i is constructed using $H_0, c_1, ..., c_i$, the probability of $t_{i+1} - t_i \geq \frac{n}{16m}$ under the condition of $H_0, c_1, ..., c_i$ is equivalent to the probability under the condition of H_i.

Given our randomly sampled challenge $j \in \{0, \ldots, n-1\}$, we let $t_{min} := \min_{v\in B, v\leq j}\{j - v\}$ denote the time cost of computing X_j given only the labels X_v for each $v \in B$. Then for any $H_i \notin \texttt{COLLISION}_i \cup \texttt{WRONGORDER}_i \cup \texttt{PREDICTABLE}$, either $\sum_{j=t_i}^{t_{i+1}-1} \texttt{NBits}(S_j, R_j) > w/4$ or $t_{i+1} - t_i \geq t_{min}$. We will show that for all $H_0, c_1, ..., c_i$ such that $H_i \notin \texttt{COLLISION}_i \cup \texttt{WRONGORDER}_i \cup \texttt{PREDICTABLE}$, we have

$$\Pr\left[t_{min} \geq \frac{n}{16m} \,\middle|\, H_0, c_1, ..., c_i\right] \geq \frac{1}{2}\,,$$

where the probability is taken over the selection of c_{i+1}. We conclude that $\Pr[t_{i+1} - t_i \geq \frac{n}{16m} \vee \sum_{j=t_i}^{t_{i+1}-1} \texttt{NBits}(S_j, R_j) \geq w/4 \,|\, H_0, c_1, ..., c_i] \geq \frac{1}{2}$.

Note that $|B_i| < 8m$ for all $H_i \notin \texttt{COLLISION}_i \cup \texttt{WRONGORDER}_i \cup \texttt{PREDICTABLE}$, so we can bound the probability as $\Pr\left[t_{min} \geq \frac{n}{16m} \,\middle|\, H_0, c_1, ..., c_i\right] \geq \Pr\left[t_{min} \geq \frac{n}{2(|B_i|+1)} \,\middle|\, H_0, c_1, ..., c_i\right]$. We denote the elements in B_i to be $b_1, ..., b_{|B_i|}$, where $b_1 < \cdots < b_{|B_i|}$. Given the labels that appear as inputs to the random oracle before appearing as outputs, we partition the label indices into $|B_i| + 1$ intervals: $[0, b_1), [b_1, b_2), ..., [b_{|B_i|}, n)$. Let $b_0 = 0, b_{|B_i|+1} = n$. Then for each challenge index c in the interval $[b_k, b_{k+1})$ for $0 \leq k \leq |B_i|$, if $\frac{n}{2(|B_i|+1)} \geq b_{k+1} - b_k$, then the attacker can compute any challenge X_c using time less than $\frac{n}{2(|B_i|+1)}$; otherwise, the attacker needs $c - b_k \geq \frac{n}{2(|B_i|+1)}$ time to compute challenge X_c for $c \in \left[b_k + \frac{n}{2(|B_i|+1)}, b_{k+1}\right)$. This means, for each interval $[b_k, b_{k+1})$ ($k = 0, 1, ..., |B_i|$), there are $\max\left(0, b_{k+1} - b_k - \frac{n}{2(|B_i|+1)}\right)$ challenges that need at least $\frac{n}{2(|B_i|+1)}$ time to compute. Therefore, we have:

$$\Pr\left[t_{min} \geq \frac{n}{16m} \,\middle|\, H_0, c_1, ..., c_i\right] \geq \Pr\left[t_{min} \geq \frac{n}{2(|B_i|+1)} \,\middle|\, H_0, c_1, ..., c_i\right]$$
$$= \frac{\sum_{k=0}^{|B_i|} \max\left(0, b_{k+1} - b_k - \frac{n}{2(|B_i|+1)}\right)}{n} \geq \frac{\sum_{k=0}^{|B_i|}\left(b_{k+1} - b_k - \frac{n}{2(|B_i|+1)}\right)}{n} = \frac{n - 0 - (|B_i|+1)\frac{n}{2(|B_i|+1)}}{n} = \frac{1}{2}$$

Since for each case of $t_{min} \geq \frac{n}{16m}$, either $t_{i+1} - t_i \geq t_{min}$ or $\sum_{j=t_i}^{t_{i+1}-1} \texttt{NBits}(S_j, R_j) > w/4$, we have $\Pr\left[t_{i+1} - t_i \geq \frac{n}{16m} \vee \sum_{j=t_i}^{t_{i+1}-1} \texttt{NBits}(S_j, R_j) \geq w/4 \,\middle|\, H_0, c_1, ..., c_i\right] \geq \Pr\left[t_{min} \geq \frac{n}{16m} \mid H_0, c_1, ..., c_i\right] \geq \frac{1}{2}$. □

Indicator L_i. For $i < n$, let $L_i \in \{0, 1\}$ be an indicator random variable for the i^{th} partial execution trace $\mathsf{Trace}_{\mathcal{A},H,i}(x)$. In particular, we set $L_i = 1$ if $\mathsf{Trace}_{\mathcal{A},H,i}(x)$ is not "lucky" given random oracle H_i *or* $H_i \in \texttt{COLLISION}_i \cup \texttt{WRONGORDER}_i \cup \texttt{PREDICTABLE}$; otherwise $L_i = 0$. Next, we will first bound the probability of $L_i = 1$ for each $i < n$, and then use concentration bounds to show that $\sum_{i=0}^{n-1} L_i \geq \frac{n}{4}$ is true for most random oracles.

Lemma 6.7. *For any $i < n$, $\Pr[L_i = 1 \mid L_0, ..., L_{i-1}] \geq \frac{1}{2}$.*

Proof. Consider an random oracle H_i constructed uniformly at random using $H_0, c_1, ..., c_i$.

If $H_i \in \texttt{COLLISION}_i \cup \texttt{WRONGORDER}_i \cup \texttt{PREDICTABLE}$, $\Pr[L_i = 1 \mid L_0, ..., L_{i-1}] = \Pr[L_i = 1] = 1$.

If $H_i \notin \texttt{COLLISION}_i \cup \texttt{WRONGORDER}_i \cup \texttt{PREDICTABLE}$, $L_i = 0$ means $\mathsf{Trace}_{\mathcal{A},H,i}(x)$ is a lucky partial trace; $L_i = 1$ means $\mathsf{Trace}_{\mathcal{A},H,i}(x)$ is not "lucky". Note that if $\mathsf{Trace}_{\mathcal{A},H,i}(x)$ is "lucky", then at most $w/4$ bits are transferred between memory and cache while $t_{i+1} - t_i \leq \frac{c_b}{4c_r}$. Also, we assume $\frac{n}{4m} \cdot c_r > c_b$ at the beginning of this section. Then by Lemma 6.6 we can bound the probability of $L_i = 1$ as below:

$$\Pr[L_i = 1 \mid H_0, c_1, ..., c_i] \geq \Pr\left[t_{i+1} - t_i \geq \frac{c_b}{4c_r} \vee \sum_{j=t_i}^{t_{i+1}-1} \texttt{NBits}(S_j, R_j) \geq w/4 \,\middle|\, H_0, c_1, ..., c_i\right]$$

$$> \Pr\left[t_{i+1} - t_i \geq \frac{n}{16m} \vee \sum_{j=t_i}^{t_{i+1}-1} \texttt{NBits}(S_j, R_j) \geq w/4 \,\middle|\, H_0, c_1, ..., c_i\right] \geq \frac{1}{2}$$

Define $\mathcal{H}_{consistent}$ as the set of all $\{H_0, c_1, \ldots, c_i\}$ consistent with $L_0, \ldots, L_{i-1}$. Then we have:

$$\Pr[L_i = 1 \mid L_0, ..., L_{i-1}] \geq \min_{\{H_0, c_1, \ldots, c_i\} \in \mathcal{H}_{consistent}} \Pr[L_i = 1 \mid H_0, c_1, ..., c_i] \geq \frac{1}{2}$$

□

Next, we will use concentration bounds to show that $\sum_{i=0}^{n-1} L_i \geq \frac{n}{4}$ is true for most random oracles.

Lemma 6.8.

$$\Pr\left[\sum_{i=0}^{n-1} L_i < \frac{n}{4}\right] < \exp\left(-\frac{n}{8}\right)$$

where the probability is taken over the choice of random oracle H with an arbitrary fixed input X.

Proof. Lemma 6.7 proves that $\Pr[L_i = 1 \mid L_0, ..., L_{i-1}] \geq \frac{1}{2}$ for any $i < n$. Noting that the random variables $L_0, ..., L_{n-1}$ are not independent, we define independent Bernoulli random variables $L'_0, ..., L'_{n-1}$ with $\Pr[L'_i = 1] = \frac{1}{2}$. Then $\Pr[L'_i = 1] \leq \Pr[L_i = 1 \mid L_0, ..., L_{i-1}]$ for all $i < n$. Thus, we have $\Pr\left[\sum_{i=0}^{n-1} L_i < \frac{n}{4}\right] \leq \Pr\left[\sum_{i=0}^{n-1} L'_i < \frac{n}{4}\right]$. Using Chernoff bound we have:

$$\Pr\left[\sum_{i=0}^{n-1} L_i < \frac{n}{4}\right] \leq \Pr\left[\sum_{i=0}^{n-1} L'_i < \mathbb{E}\left[\sum_{i=0}^{n-1} L'_i\right] - \frac{n}{4}\right] < \exp\left(-\frac{n}{8}\right).$$

An alternate way to prove this lemma is to define a sequence $X_k = \sum_{i=1}^{k} L_i - \frac{k+1}{2}$ for $k = 0, \ldots, n-1$, observe that the sequence can be viewed as a submartingale, and then apply Azuma's inequality to bound X_{n-1}. □

Denote E_1 be the set of random oracles such that $\sum_{i=0}^{n-1} L_i \geq \frac{n}{4}$, $\mathtt{SUCCESS}$ be the set of random oracles such that $\mathsf{Trace}_{\mathcal{A},H}(x)$ outputs Y_n correctly for $H \in \mathtt{SUCCESS}$. Note that $|\overline{E_1}| < \mathcal{S} \cdot \exp\left(-\frac{n}{8}\right)$, and $|\mathtt{SUCCESS}| \geq \mathcal{S}\epsilon$. Then $E_1 \cap \mathtt{SUCCESS} \cap \overline{\mathtt{COLLISION}_n} \cap \overline{\mathtt{WRONGORDER}_n} \cap \overline{\mathtt{PREDICTABLE}}$ is the set of random oracles such that for each $H \in E_1 \cap \mathtt{SUCCESS} \cap \overline{\mathtt{COLLISION}_n} \cap \overline{\mathtt{WRONGORDER}_n} \cap \overline{\mathtt{PREDICTABLE}}$, there are at least $\frac{n}{4}$ of the partial execution traces in the entire trace each of which costs no less than $\frac{c_b}{4}$ energy, and thus $\mathsf{cost}(\mathsf{Trace}_{\mathcal{A},H}(x)) \geq \sum_{i:L_i=1} \mathsf{cost}(\mathsf{Trace}_{\mathcal{A},H,i}(x)) \geq \frac{n}{4} \cdot \frac{c_b}{4} = \frac{nc_b}{16}$.

In the end, we only need to bound the probability of $H \in E_1 \cap \mathtt{SUCCESS} \cap \overline{\mathtt{COLLISION}_n} \cap \overline{\mathtt{WRONGORDER}_n} \cap \overline{\mathtt{PREDICTABLE}}$ to finish the proof of Theorem 6.2.

Note that $E_1 \cap \mathtt{SUCCESS} \cap \overline{\mathtt{COLLISION}_n} \cap \overline{\mathtt{WRONGORDER}_n} \cap \overline{\mathtt{PREDICTABLE}} \geq |\mathtt{SUCCESS}| - |\overline{E_1}| - |\mathtt{COLLISION}_n| - |\mathtt{WRONGORDER}_n| - |\mathtt{PREDICTABLE}| \geq \mathcal{S} \cdot (\epsilon - \exp\left(-\frac{n}{8}\right) - \frac{3}{2}n^3 2^{-w} - qn^2 2^{-w} - 2^{-mw/5})$. Since H is chosen uniformly at random, we have:

$$\begin{aligned}&\Pr\left[H \in E_1 \cap \mathtt{SUCCESS} \cap \overline{\mathtt{COLLISION}_n} \cap \overline{\mathtt{WRONGORDER}_n} \cap \overline{\mathtt{PREDICTABLE}}\right] \\ \geq &\epsilon - \exp\left(-\frac{n}{8}\right) - \frac{3}{2}n^3 2^{-w} - qn^2 2^{-w} - 2^{-mw/5}.\end{aligned}$$

This completes the proof of Theorem 6.2.

Acknowledgements

The authors would like to thank Daniel Wichs for helpful discussion and anonymous reviewers for important comments that improved the presentation of the paper. The research was supported in part by the National Science Foundation under NSF Awards #1704587 and #1649515 and #1755708 and #2047272 and by a gift from Protocol Labs. The views expressed in this paper are those of the authors and do not necessarily reflect those of the National Science Foundation or Protocol Labs. This work was done in part while Samson Zhou was affiliated with Purdue University and Carnegie Mellon University.

References

[1] Martín Abadi, Michael Burrows, Mark S. Manasse, and Ted Wobber. Moderately hard, memory-bound functions. *ACM Trans. Internet Techn.*, 5(2):299–327, 2005. 1

[2] Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 241–271. Springer, Heidelberg, August 2016. 1, 4, 1.2, 2.1, 4, 4

[3] Joël Alwen and Jeremiah Blocki. Towards practical attacks on argon2i and balloon hashing. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 142–157. IEEE, 2017. 1, 2, 4

[4] Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical graphs for optimal side-channel resistant memory-hard functions. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1001–1017. ACM Press, October / November 2017. 1, 1.2, 4, 2.1, 5, 5.2, 5.4, 5.5, 1, 2

[5] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 3–32. Springer, Heidelberg, April / May 2017. 1, 4, 2.1, 4

[6] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 33–62. Springer, Heidelberg, April / May 2017. 1, 1.2, 1.2, 4, 5.3, 6.1, 6, 6.3, 6.4, 6, 6, B.2

[7] Joël Alwen, Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. Cumulative space in black-white pebbling and resolution. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, 9-11 January 2017, Berkeley, California USA*, 2016. 1.2, 4, 4

[8] 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. 1, 1.1, 1.2, 1.2, 3, 3.2, 3.4, 4, 11

[9] Adam Back. Hashcash-a denial of service counter-measure, 2002. 1

[10] Daniel J. Bernstein. Cache-timing attacks on aes, 2005. 1

[11] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Fast and tradeoff-resilient memory-hard functions for cryptocurrencies and password hashing. *IACR Cryptology ePrint Archive*, 2015:430, 2015. 5, 3, 4

[12] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 292–302, 2016. 1, 1.2, 5.2

[13] Alex Biryukov, Daniel Dinu, Dmitry Khovratovich, and Simon Josefsson. The memory-hard argon2 password hash and proof-of-work function, March 2016. 1.2, 3

[14] Jeremiah Blocki, Benjamin Harsha, Siteng Kang, Seunghoon Lee, Lu Xing, and Samson Zhou. Data-independent memory hard functions: New attacks and stronger constructions. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 573–607. Springer, Heidelberg, August 2019. 4, 5.4

[15] Jeremiah Blocki, Benjamin Harsha, and Samson Zhou. On the economics of offline password cracking. In *2018 IEEE Symposium on Security and Privacy*, pages 853–871. IEEE Computer Society Press, May 2018. 1

[16] Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of Argon2i. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 445–465. Springer, Heidelberg, November 2017. 1, 2.1, 5.3

[17] Dan Boneh, Henry Corrigan-Gibbs, and Stuart E. Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 220–248. Springer, Heidelberg, December 2016. 1, 5.2

[18] Erik D. Demaine and Quanquan C. Liu. Inapproximability of the standard pebble game and hard to pebble graphs. In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, pages 313–324, 2017. 1.2

[19] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO, 12th Annual International Cryptology Conference, Proceedings*, pages 139–147, 1992. 1

[20] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. Key-evolution schemes resilient to space-bounded leakage. In *Advances in Cryptology - CRYPTO - 31st Annual Cryptology Conference, Proceedings*, pages 335–353, 2011. 3.1

[21] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC Proceedings*, pages 125–143, 2011. 3.1

[22] Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena: A memory-consuming password scrambler. Cryptology ePrint Archive, Report 2013/525, 2013. https://eprint.iacr.org/2013/525. 1

[23] John R. Gilbert, Thomas Lengauer, and Robert Endre Tarjan. The pebbling problem is complete in polynomial space. In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing (STOC)*, pages 237–248, 1979. 1.1, 1.2, 6, C, C, C.1, C.1, D

[24] Jia-Wei Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 326–333, 1981. 1.1

[25] Thomas Lengauer and Robert E. Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *J. ACM*, 29(4):1087–1130, October 1982. 1

[26] Quanquan Liu. Red-blue and standard pebble games: Complexity and applications in the sequential and parallel models. Master's thesis, Massachusetts Institute of Technology, Feburary 2017. 1.2, D

[27] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. 1

[28] Colin Percival. Stronger key derivation via sequential memory-hard functions. *BSDCan*, 2009. 1, 1.2, 6

[29] Password hashing competition, 2013–2015. 1, 1.2

[30] Ling Ren and Srinivas Devadas. Bandwidth hard functions for ASIC resistance. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 466–492. Springer, Heidelberg, November 2017. 1, 1.2, 1.2, 1.2, 6

[31] Georg Schnitger. On depth-reduction and grates. In *24th Annual Symposium on Foundations of Computer Science*, pages 323–328, 1983. 1.2, 4, 4

[32] Craig A. Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. D

A Specification of Candidate iMHFs

In this section we give provide detailed descriptions of the iMHFs analyzed in the main body of the paper. DRSample is described in Algorithm 1, aATSample is described in Algorithm 2, Argon2iB is described in Algorithm 3 and Argon2iA is described in Algorithm 4. The aATSample construction in Algorithm 2 uses DRSample (Algorithm 1) as a building block. Intuitively, the subgraph induced by the first $n/2$ nodes form a DRSample graph with $n/2$ nodes and the following $n/2$ nodes form a path with additional parents selected from DRSample.

B Missing Proofs

Reminder of Theorem 5.2. *Let $G = ([n], E)$ be any DAG such that $(j, j+1) \in E$ for each $j < n$, let c be a positive integer and let $T_i = ((i-1)c\ell + 1, ic\ell]$,*

$$\mathsf{rbpeb}^{\parallel}(G, m) \geq \sum_{i=1}^{\left\lfloor \frac{n}{c\ell} \right\rfloor} \min_{R, B' \subseteq [(i-1)c\ell] : |R| \leq m} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T_i)|\, c_r \right) .$$

Algorithm 1: An algorithm for sampling depth-robust graphs. [4]

```
Function DRSample($n \in \mathbb{N}_{\geq 2}$):
    $V := [v]$
    $E := \{(1,2)\}$
    for $v \in [3,n]$ and $i \in [2]$ do                    // Populate edges
        $E := E \cup \{(v, \mathsf{GetParentDRS}(v,i))\}$    // Get $i^{th}$ parent
    end
    return $G := (V,E)$.

Function GetParentDRS($v$,$i$):
    if $i = 1$ then
        $u := i - 1$
    else
        $g' \leftarrow [1, \lfloor \log_2(v) \rfloor + 1]$   // Get random range size.
        $g := \min(v, 2^{g'})$                              // Don't make edges too long.
        $r \leftarrow [\max(g/2, 2), g]$                     // Get random edge length.
    end
    return $v - r$
```

Proof of Theorem 5.2: (Sketch) Repeatedly invoke Lemma 5.1. Consider an optimal red-blue pebbling and let t_i denote the first time we place a pebble on node $ic\ell$. For each i the red-blue cost incurred between steps $t_{i-1}+1$ and t_i starting from some red-blue configuration $B_{t_{i-1}}, R_{t_{i-1}}$ is at least

$$\begin{aligned}
&\mathsf{rbpeb}^{\parallel}(G, m, T_i, B_{t_{i-1}}, R_{t_{i-1}}) \\
&\quad\geq \min_{B' \subseteq [(i-1)c\ell]} \left(|B'|\, c_b + \left|\mathsf{ancestors}_{G-R_{t_{i-1}}-B'}(T_i)\right| c_r \right) \\
&\quad\geq \min_{R,B' \subseteq [(i-1)c\ell] : |R| \leq m} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T_i)|\, c_r \right) .
\end{aligned}$$

To complete the proof we observe that

$$\mathsf{rbpeb}^{\parallel}(G,m) \geq \sum_{i=1}^{\lfloor \frac{n}{c\ell} \rfloor} \mathsf{rbpeb}^{\parallel}(G, m, T_i, B_{t_{i-1}}, R_{t_{i-1}}) .$$

□

B.1 aATSample

Reminder of Lemma 5.6. *Let $i > \frac{n}{2}$ and $T = [i, i+\ell-1]$ be an interval of length $\ell = \frac{n}{\log n}$. Then for any parameters $c \geq 1$ and $m \leq \frac{n}{16c\log n}$ a graph generated by* aATSample(n,c) *satisfies the following property:*

$$\min_{R,B' \subseteq [i-1] : |R| \leq m} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r \right) \geq \min\left(\frac{n}{16c\log n} c_b, \frac{n}{8} c_r \right)$$

Proof of Lemma 5.6: We first consider casework on the size of B'. If $|B'| \geq \frac{n}{16c\log n}$, then we trivially have $|B'|c_b \geq \frac{n}{16c\log n} c_b$. Otherwise, we have $|B'| \leq \frac{n}{16c\log n}$, in which case $|R \cup B'| \leq \frac{n}{8c\log n}$ since $|R| \leq m$. We now lower bound $|\mathsf{ancestors}_{G-R-B'}(T)|\, c_r$ under the assumption that $|R \cup B'| < \frac{n}{8c\log n}$.

Partition the nodes $[n/2]$ into $\frac{n}{2k}$ intervals $[1,k], [k+1, 2k], \ldots$ where $k = 2c\log n$ and $c \geq 1$ is the parameter used in Algorithm 2 (aATSample). Observe that for each interval $[(v-1)k+1, vk]$ the graph G

Algorithm 2: An algorithm for sampling a high aAT graph. [4]

```
Function aATSample($n, c$):
    $V := [n]$
    $E := \{(i, i+1) \ : i \in [n-1]\}$
    for $v \in [3, n]$ and $i \in [2]$ do                    // Populate new edges of graph.
        $E := E \cup \{(v, \mathsf{GetParent}^c(v, i))\}$          // Get $i^{th}$ parent of node $v$
    end
    return $G := (V, E)$.

Function GetParent$^c$($v$,$i$):
    if $i = 1$ then
        $u := i - 1$
    end
    else if $v \leq n/2$ then
        $u := \mathsf{GetParentDRS}(v, i)$              // First $n/2$ nodes form copy of DRSample

    end
    else
        $m := \lfloor c \log(n) \rfloor$
        $b := (v - n) \mod \lfloor \frac{n}{2m} \rfloor$
        $u := bm$
    end
    return $u$
```

contains an edge from some node $x \in [vk - k/2 + 1, vk]$ (the second half of the interval) to some node $y \in T$. Let $B_k = \{v \leq \frac{n}{2k} \ : [(v-1)k+1, vk] \cap (R \cup B') \neq \emptyset\}$ denote the set of intervals which intersect with $R \cup B'$. Clearly, $|B_k| \leq |R \cup B'| \leq \frac{n}{8c \log n}$. We claim that if $v \notin B_k$ then every node in $[(v-1)k, vk - k/2]$ (first half of the interval) is also in $\mathsf{ancestors}_{G-R-B'}(T)$. To see this observe that for each interval $[(v-1)k+1, vk]$ the graph G contains an edge from some node $x \in [vk - k/2 + 1, vk]$ to some node $y \in T$ and the entire interval $[(v-1)k+1, vk]$ is disjoint from $B' \cup R$. Thus, we have at least $\frac{k}{2}\left(\frac{n}{2k} - |B_k|\right) = \frac{n}{8}$ nodes in $\mathsf{ancestors}_{G-R-B'}(T)$. □

B.2 DRSample

Reminder of Lemma 5.5. *Suppose $m = \mathcal{O}(n^{\rho})$ for some constant $0 < \rho < 1$ and $i > \frac{n}{2}$. Let $T = [i, i + \ell - 1]$ be an interval of length $\ell \geq 16m/(1-\rho)$. Then a graph generated by* DRSample *satisfies the following with high probability:*

$$\min_{R \subseteq [i-1] : |R| \leq m} \min_{B' \subseteq [i-1]} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r\right) \geq \min\left(\frac{(1-\rho)\ell}{8} c_b, \left(\frac{(1-\rho)\ell}{16}\right)\sqrt{\frac{n}{64\ell}} c_r\right)$$

Proof of Lemma 5.5:

Let $T = [i, i + \ell]$ where $\ell \geq 16m/(1-\rho)$ for some constant $\frac{1}{2} < \rho < 1$ and let $r(j)$ denote the predecessor of a node j in the graph (besides $j-1$) i.e., $r(j) = \texttt{GetParent}(j, 2)$. We first note that if $|B'| \geq \frac{c\ell}{2}$ for the constant $c = \frac{1-\rho}{4}$ then $|B'| c_b \geq \frac{c\ell}{2} c_b$ and we are immediately done. Otherwise, we let $b = \sqrt{\frac{n}{64\ell}}$ and for $i < j \leq i + \ell$, let X_j be an indicator random variable for the event $\mathsf{far}(j)$, which we define to be the event that $|r(j) - r(k)| > b$ for all $k \in [i, j-1]$ and $r(j) < i$. Observe that if $\mathsf{far}(j) = 1$ then either $B' \cup R$ contains some node in the interval $[r(j) - b, r(j)]$ or these nodes will be contained in $\mathsf{ancestors}_{G-R-B'}(T)$. In particular, if $X = \sum_{i \in T} X_i$ we have $\mathsf{ancestors}_{G-R-B'}(T) \geq (X - m - |B'|)$. It remains to lower bound

Algorithm 3: An algorithm for sampling depth-robust graphs. [11]

```
Function Argon2iB($n \in \mathbb{N}_{\geq 2}$):
    $V := [v]$
    $E := \{(1,2)\}$
    for $v \in [3,n]$ and $i \in [2]$ do                              // Populate edges
        $E := E \cup \{(v, \mathsf{GetParent}(v,i))\}$                // Get $i^{th}$ parent
    end
    return $G := (V,E)$.

Function GetParent($v$,$i$):
    if $i = 1$ then
        $u := i - 1$
    else
        $N := 2^{32}$                                                 // Set sample range.
        $g \leftarrow [1,N]$                                          // Get random range length.
        $r := \left[\frac{g^2}{N^2} v\right]$                          // Set quadratic dependency.
    end
    return $v - r$
```

X. Observe that for any setting of $r(i), \ldots, r(j-1)$ the set $S = \bigcup_{y=i}^{j-1}[r(y)-b, r(y)+b]$ has size at most $(2b+1)(j-i)$ and thus $\Pr[r(j) \in S]$ is maximized when $S = [i-(2b+1)(j-i), i-1]$. Hence,

$$\begin{aligned}
\mathbf{Pr}\left[\mathsf{far}(j)\right] &\geq \mathbf{Pr}\left[r(j) < i - (j-i)(2b+1)\right] \\
&\geq \mathbf{Pr}\left[j - r(j) > \ell + (j-i)(2b+1)\right] \\
&\geq \mathbf{Pr}\left[j - r(j) > \ell + (\ell)(2b+1)\right] \\
&\geq \mathbf{Pr}\left[j - r(j) > \frac{\sqrt{n\ell}}{2}\right]
\end{aligned}$$

since $j \leq i + \ell$ and $b = \sqrt{\frac{n}{64\ell}}$. In the last inequality we assume that $n \geq 64\ell$ so that $b \geq 1$ and $\ell + \ell(2b+1) \leq 4\ell b = \frac{\sqrt{n\ell}}{2}$. Hence,

$$\begin{aligned}
\mathbf{Pr}\left[\mathsf{far}(j)\right] &\geq \frac{\log(j) - \log\sqrt{n\ell}}{\log(j)} \\
&\geq 1 - \left(\frac{1}{2} - \frac{\rho}{2}\right)\left(\frac{\log(n)}{\log(n)-1}\right) \\
&\geq \frac{1}{2} - \frac{\rho}{2} - o(1) = \Omega(1).
\end{aligned}$$

Let $c = \frac{1-\rho}{4}$ With high probability, $X = \sum_{k=i}^{i+\ell} X_k > c\ell$. Setting $\ell \geq 4m/c$, then with high probability, the number of ancestors of T in $G - R - B'$ is at least

$$\begin{aligned}
(X - |R| - |B'|)b &\geq (X - m - |B'|)b \\
&\geq \left(\frac{c\ell}{4}\right)\sqrt{\frac{n}{64\ell}},
\end{aligned}$$

for $|B'| \leq \frac{c\ell}{2}$. Thus, either $|B'| \geq \frac{c\ell}{2}$ or $|\mathsf{ancestors}_{G-R-B'}(T)| \geq \left(\frac{c\ell}{4}\right)\sqrt{\frac{n}{64\ell}}$. It follows that

$$\min_{R \subseteq [i-1]:|R| \leq m} \min_{B' \subseteq [i-1]} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r\right) \geq \min\left(\frac{c\ell}{2} c_b, \left(\frac{c\ell}{4}\right)\sqrt{\frac{n}{64\ell}} c_r\right).$$

Algorithm 4: An algorithm for sampling depth-robust graphs. [11]

```
Function Argon2iA($n \in \mathbb{N}_{\geq 2}$):
    $V := [v]$
    $E := \{(1,2)\}$
    for $v \in [3,n]$ and $i \in [2]$ do                 // Populate edges
        $E := E \cup \{(v, \mathsf{GetParent}(v,i))\}$    // Get $i^{th}$ parent
    end
    return $G := (V,E)$.

Function GetParent($v$,$i$):
    if $i = 1$ then
        $u := i - 1$
    else
        $N := 2^{32}$                                     // Set sample range.
        $g \leftarrow [1,N]$                              // Get random range length.
        $r := \lceil \frac{g}{N} v \rceil$                // Set linear dependency.
    end
    return $v - r$
```

□

We now give an alternate bound for DRSample when the cache has size $\mathcal{O}(n^\rho / \log n)$ for any $0 < \rho < 1$. It shows that either the pebbling has $\tilde{\Omega}(n)$ blue moves or there are at least $\tilde{\Omega}(n^{2-\rho})$ red moves. The alternate bound is incomparable to our prior bound showing that any pebbling either has $\Omega(n)$ blue moves or at least $\Omega(n^{3/2-3\rho/2})$ red moves. In particular, we cannot minimize the number of blue moves without paying a steep cost in the number of red moves.

Lemma B.1. *Suppose $m = Cn^\rho/\log n$ for some constants $C > 0$ and $0 < \rho \leq 1$ and $i > \frac{n}{2}$. Let $T = [i, i+\ell-1]$ be an interval of length $\ell = 100 m \log n$. Then a graph generated by* DRSample *satisfies the following with high probability:*

$$\min_{R \subseteq [i-1]: |R| \leq m} \ \min_{B' \subseteq [i-1]} \left(|B'|\, c_b + |\mathsf{ancestors}_{G-R-B'}(T)|\, c_r \right) \geq \min\left(m c_b, \frac{n}{24} \cdot c_r \right).$$

Proof. Let T be an interval of length ℓ. If $|B'| > m$ then we immediately have $|B'|\, c_b > m c_b$. Thus, in the remainder of the proof we assume that $|B'| \leq m$ so that $|R \cup B'| \leq 2m$. Partition nodes in G into intervals $I_1, I_2, \ldots$ of length $k = \frac{n}{12m} = \mathcal{O}\left(n^{1-\rho} \log n\right)$ where $I_j = [(j-1)k+1, jk]$. Let $L_j = [(j-1)k + \lceil k/2 \rceil + 1, jk]$ (resp. $F_j = [(j-1)k+1, (j-1)k + \lceil k/2 \rceil]$) denote the last (resp. first) half of the nodes in I_j. Now for each $j \in T$ define the random variable $X_j = 1$ if for some $i' \leq \frac{n}{2k}$ we have $r(j) \in L_{i'}$ and for all prior nodes $i \leq j' < j$ in the interval T we have $r(j') \notin E_{i'}$; otherwise $X_j = 0$. Intuitively, $X_j = 1$ if the edge $r(j)$ is connected to (the second half of) a new interval. Let $B_k = \{i' \ : \ |I_{i'} \cap (B' \cup R)| \geq 1\}$ be the set of intervals that contain some node in $B' \cup R$ and let $X = \sum_{j \in T} X_j$. Observe that there are at least $X - |B_k| - m \geq X - 2Cn^{1-\epsilon}$ intervals $I_{i'}$ such that (1) the interval $I_{i'}$ contains no node in $B' \cup R$ i.e., $I_{i'} \cap (B' \cup R) = \{\}$, and (2) there is an edge $(r(j), j)$ with $j \in T$ and $r(j) \in L_{i'}$. For each such interval $I_{i'}$ the entire interval $F_{i'}$ is contained in $\mathsf{ancestors}_{G-R-B'}(T)$ because the graph G contains all directed edges of the form $(i, i+1)$ for $i < n$.

Thus,

$$|\mathsf{ancestors}_{G-R-B'}(T)| \geq (X - 2m)\, \frac{k}{2} .$$

We now argue that $X \geq \min\{\frac{n}{4k}, \frac{\ell}{25 \log n}\}$ with high probability. To see this observe that if $X_1 + \ldots + X_{j-1} \leq \frac{n}{4k}$ then there at least $\frac{n}{4k}$ of the intervals $I_1, \ldots I_{\frac{n}{2k}}$ are still "uncovered" and for each uncovered interval $I_{i'}$ we have

$$\Pr[r(j) \in F_{i'}] \geq \frac{k}{2n \log n} \ .$$

Thus, we have

$$\Pr\left[X_j = 1 \middle| X_1 + \ldots + X_{j-1} \leq \frac{n}{4k}\right] \geq \frac{k}{2n \log n} \times \frac{n}{4k} \geq \frac{1}{8 \log n} \ .$$

Thus, in expectation we have $\mathbb{E}[X] \geq \min\{\frac{n}{4k}, \frac{\ell}{8 \log n}\}$. We picked our parameters such that $\frac{n}{4k} = 3m$ and $\frac{\ell}{25 \log n} = 4m$. We can apply concentration bounds to argue that (whp) we have $X \geq \frac{\ell}{4k} \geq 3m$. To see this we can introduce new random variables Y_j such that $Y_j = 1$ if either $X_j = 1$ or $X_1 + \ldots + X_{j-1} \geq \frac{n}{4k}$. By definition, we have $\sum_{j \in T} Y_j \geq \frac{n}{4k}$ if and only if $\sum_{j \in T} X_j \geq \frac{n}{4k}$. We also have $\Pr[Y_j = 1 \ | Y_i = y_i, \ldots Y_{j-1} = y_{j-1}] \geq \frac{1}{8 \log n}$ for all prior outcomes $y_i, \ldots, y_{j-1} \in \{0,1\}$. We can apply concentration bounds to upper bound $\Pr[Y \leq \frac{n}{4k}]$ (e.g., see Generalized Hoeffding Inequality [6, Claim 7]) because $\Pr[Y_j = 1 \mid (Y_i, \ldots, Y_{j-1}) = (y_i, \ldots, y_{j-1})] \geq \frac{1}{8 \log n}$ for *all* prior outcomes $y_i, \ldots, y_{j-1} \in \{0,1\}$. It follows that (whp) $X - 2m \geq m$ and

$$|\mathsf{ancestors}_{G-R-B'}(T)| \geq m\frac{k}{2} = \frac{n}{24} \ .$$

□

Theorem B.2. *Let G be a graph generated by* DRSample *and $0 < \rho \leq 1$. Then there exists a constant $C > 0$ so that for all $m \leq Cn^\rho / \log n$, it follows that*

$$\mathsf{rbpeb}^{\parallel}(G, m) \geq C \cdot \min(\frac{n}{\log n} c_b, \frac{n^2}{m \log n} c_r)$$

with high probability.

Proof. Applying Lemma B.1 to each of the disjoint $\frac{n}{\ell} = \frac{n}{100m \log n}$ intervals in the second half of graph G and observing that $\ell = \mathcal{O}(n^\rho)$, it follows from Theorem 5.2 that

$$\mathsf{rbpeb}^{\parallel}(G, m) \geq \min(\Omega(n/\log n)c_b, \Omega(n^2/(m \log n))c_r).$$

□

We remark that if $m = o(n/\log n)$ in Theorem B.2, e.g., $m = n/(\log n \log \log n)$, then we have $\frac{n^2}{m} = \omega(nc_r)$ and $\mathsf{rbpeb}^{\parallel}(G, m) \geq C \cdot \min(\frac{n}{\log n} c_b, \omega(nc_r))$.

B.3 Argon2i Edge Distribution

Reminder of Lemma 5.3. *Let G be a random Argon2iB (resp. Argon2iA) graph with n nodes then for any $1 \leq j < i - 1 \leq n$ we have $\Pr[r(i) = j] \geq \frac{1}{3n}$ (resp. $\Pr[r(i) = j] \geq \frac{1}{n}$).*

Proof of Lemma 5.3: Let $1 \leq j < i - 1 < n$ be given. For Argon2iA the edge distribution for $r(i)$ is uniform over the set $\{1, \ldots, i-2\}$ so for any $j \leq i - 2$ we have $\Pr[r(i) = j] = \frac{1}{i-2} \geq \frac{1}{n}$. In the Argon2iB edge distribution to determine the value $r(i) < i - 1$ for the directed edge $(r(i), i)$ we have

$$\Pr[r(i) = j] = \Pr_{x \in [N]}\left[i\left(1 - \frac{x^2}{N^2}\right) \in (j-1, j]\right]$$

where $N \geq 6n$ and the randomness is taken over the selection of $x \in [N]$. Equivalently, $r(i) = j$ whenever

$$(i - j + 1)\frac{N^2}{i} \geq x^2 \geq (i - j)\frac{N^2}{i} \ .$$

The above probability is minimized when $j = 1$ and $i = n$. Thus, it suffices to lower bound $\Pr[r(n) = 1] \geq \frac{1}{3n}$. Observe that

$$\begin{aligned}
\Pr[r(n) = 1] &= \Pr\left[(i-j+1)\frac{N^2}{i} \geq x^2 \geq (i-j)\frac{N^2}{i}\right] \\
&= \Pr\left[N \geq x \geq N\sqrt{(n-1)/n}\right] \\
&= \frac{N - \lceil N\sqrt{(n-1)/n}\rceil}{N} \\
&\geq \frac{N - N\sqrt{(n-1)/n} - 1}{N} \\
&\geq \left(1 - \sqrt{\frac{n-1}{n}} - \frac{1}{6n}\right) \\
&\geq \frac{1}{3n} \ .
\end{aligned}$$

The last line follows because $1 - \frac{1}{2n} \geq \sqrt{\frac{n-1}{n}}$ i.e.,

$$\left(1 - \frac{1}{2n}\right)^2 = 1 - \frac{1}{n} + \frac{1}{4n^2} \geq \frac{n-1}{n} \ .$$

□

C Background on the Gilbert *et al.* Black Pebbling Reduction

Gilbert *et al.* [23] showed that the minimum space black pebbling problem was $\mathsf{PSPACE-Hard}$ by reduction from the Truly Quantified Boolean Formula (TQBF) problem. They provide a construction from any instance of TQBF to a DAG G_{TQBF} with pebbling number $3n+3$ if and only if the instance is satisfiable, where the pebbling number of a DAG G is $\min_{P=(P_1,\ldots,P_t)\in\mathcal{P}^{\parallel}} \max_{i\leq t} |P_i|$, the number of pebbles necessary to pebble G. For our purposes it will be sufficient to describe how their reduction map `3-SAT` instance ϕ to a DAG G_ϕ (observe that a `3-SAT` instance can be viewed as a TQBF instance in which all of the quantifiers are existential).

An important gadget in their construction is the so-called pyramid DAG, whose key property is that *any* legal pebbling of a k-pyramid requires at least k pebbles on the DAG at some point in time. A k-pyramid consists of $\sum_{i=1}^{k} i$ nodes, including k sources and a unique sink node. Formally, a pyramid graph Δ_k has nodes $V = \{v_{i,j} \ : \ 1 \leq j \leq k, 1 \leq i \leq k-j+1\}$ with k sources $v_{i,1}$ for $i \leq k$ and one sink node $v_{k,k}$. The edge set is defined as $E = \{(v_{i,j}, v_{i,j+1}) \ : \ k > j \geq 1\} \cup \{(v_{i,j}, v_{i,j+1}) \ : \ 1 \leq j < k, i < k-j+1\}$. We use both Δ_k and a triangle with the number k inside to denote a k-pyramid (see Figure 2 for an example of a 3-pyramid). The space complexity of Δ_k is exactly k.

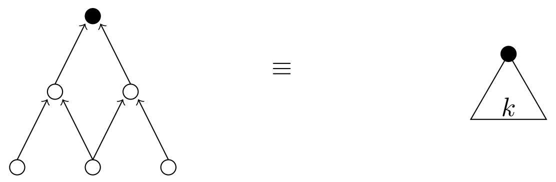

Fig. 2: A 3-Pyramid.

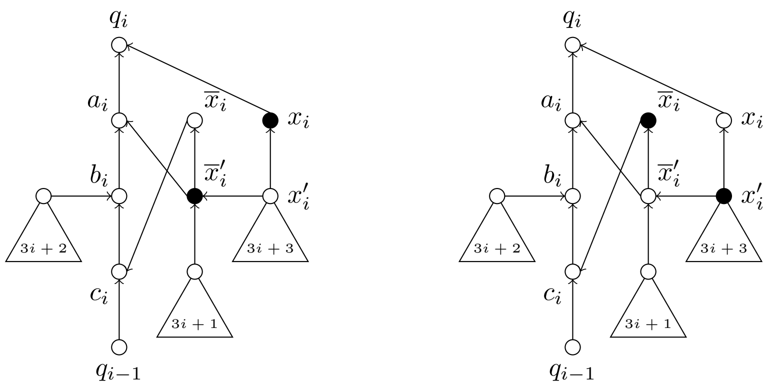

Fig. 3: A variable gadget G_{x_i} with x_i set to "true" (left figure) and x_i set to "false" (right figure). The node q_{i-1} actually belongs to $G_{x_{i-1}}$. It is drawn here to illustrate how variable gadgets are connected.

Construction of G_ϕ. Consider a `3-SAT` formula ϕ with variables $x_1, \ldots, x_n$ and 3CNF clauses $C_1, \ldots, C_c$. For each variable x_i, there is a variable gadget G_{x_i} and for each clause C_j, there is a clause gadget G_{C_j}. Each clause gadget has a sink node p_j that is connected to one of the source nodes in $G_{C_{j+1}}$, and there is a special source node p_0 that is connected to one of the source nodes in G_{C_1}. The variable gadget G_{x_i} is shown in Figure 3. This gadget in turn is constructed from three pyramid graphs $\Delta_{3i+1}, \Delta_{3i+2}$ and Δ_{3i+3}. The remaining nodes in G_{x_i} are $x_i, x'_i, \overline{x}'_i, \overline{x}_i, a_i, b_i$ and q_i. While the node c_i is a source node in G_{x_i}, it will not be a source node in the final graph G_ϕ since we will add the edges (q_{i-1}, c_i) for each $i > 1$ and (p_m, c_1) for $i = 1$. By contrast, the source nodes in the pyramids $\Delta_{3i+1}, \Delta_{3i+2}$ and Δ_{3i+3} will remain source nodes in the final graph G_ϕ. The graph G_ϕ contains a unique sink node q_n from the gadget G_{x_n}.

For each clause C_j, there exists a corresponding clause gadget that is a 3-pyramid with sink node p_j, as previously discussed. Suppose the three variables appearing in the clause are $y_{j,1}, y_{j,2}, y_{j,3} \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ and the three source nodes of the 3-pyramid are nodes $v_{j,1}, v_{j,2}, v_{j,3}$. Then we create incoming edges $(p_{j-1}, v_{j,1})$ and $(y_{j,1}, v_{j,1})$ to $v_{j,1}$, incoming edges $(y_{j,1}, v_{j,2})$ and $(y_{j,2}, v_{j,2})$ to $v_{j,2}$, and incoming edges $(y_{j,2}, v_{j,3})$ and $(y_{j,3}, v_{j,3})$ to $v_{j,3}$. Note that we can only pebble the clause gadget C_j if there exist pebbles on nodes $y_{j,1}, y_{j,2}, y_{j,3}$, corresponding to assignments for these variables. Finally for the final clause C_c, we create an edge between p_c and node q_0 of the variable gadget corresponding to x_1.

Any instance of TQBF in which each quantifier is an existential quantifier requires at most a quadratic number of pebbling moves. Specifically, we look at instances of `3-SAT`, such as in Figure 4. In such a graph representing an instance of `3-SAT`, the sink node to be pebbled is q_n. By design of the construction, any true statement requires exactly three pebbles for each pyramid representing a clause. On the other hand, a false clause requires four pebbles, so that false statements require more pebbles. Thus, by providing extraneous additions to the construction which force the number of pebbling moves to be a known constant, we can extract the pebbling number, given the space-time complexity. For more details, see the full description in [23].

C.1 Pebbling Strategy

Gilbert et al. [23] show that the DAG G_ϕ has pebbling number $3n+3$ if and only if ϕ is satisfiable. We outline the pebbling strategy below as this will be important to build intuition for our modified construct. We start off by placing a pebble on the sink nodes of *every* pyramid graph. The graph has $3n$ pyramid graphs $\Delta_{3n+3}, \Delta_{3n+2}, \ldots, \Delta_4$ where $\Delta_{3i+1}, \Delta_{3i+2}$ and Δ_{3i+3} are associated with the variable gadget G_{x_i}. We pebble the pyramid graphs in descending order of size i.e., we first place a pebble on the sink of Δ_{3n+3} using space $3n+3$ and $\sum_{i=1}^{3n+3} i$ sequential pebbling moves. We then discard all pebbles on Δ_{3n+3} except for the sink node and move on to pebble Δ_{3n+2} etc... After the sink of each pyramid has been pebbled we move each variable gadget to a true/false configuration as shown in Figure 3. We first slide a pebble from the sink of Δ_{3i+2} to node $\overline{x}'_i$. Next if the variable x_i is assigned to be true in the satisfying assignment we slide a pebble from node x'_i to x_i. On the other hand if x_i is assigned to be false we instead slide a pebble from node $\overline{x}'_i$ to node $\overline{x}_i$. Assuming the boolean formula is satisfiable we can now walk all the way across the clause gadgets to the node $q_0 = p_c$ without ever placing more than $3n + 3$ pebbles on the graph.

Advancing a Pebble from q_{i-1} to q_i. We will maintain the invariant that when we reach node q_{i-1} with a pebble we will have $3n - 3i + 3 + 1$ pebbles on the graph. The steps to move a pebble from q_{i-1} (the source in G_{x_i}) to q_i (the sink) depend on whether or not G_{x_i} is in the true or false configuration. If we are in the true configuration then we can place a pebble on $\overline{x}_i$ (keeping the pebble on node $\overline{x}'_i$ for the time being!) and then we slide the pebble on node q_{i-1} to node c_i followed by nodes b_i, a_i and q_{i+1}. If instead we are in the false configuration then we can start by sliding the pebble on node q_{i-1} to node c_i and then to node b_i. At this point we will need to pause to re-pebble node $\overline{x}'_i$ before we can place our pebble on node a_i. To place a pebble on node $\overline{x}'_i$ we will need to re-pebble the pyramid Δ_{3i+1}. To ensure we have enough space we can first discard all pebbles on G_{x_i} except for nodes b_i and x'_i leaving us with a total of $3(n - i) + 2$ pebbles on G_ϕ (including the 3 pebbles on G_{x_j} for each $j \geq i$). Since $3n + 3 - 3(n - i) - 2 = 3i + 1$ we have just enough available space to accomplish this task. Once we place a pebble on the sink of Δ_{3i+1} we can slide this pebble to node $\overline{x}'_i$ and then slide this pebble to a_i. Now we can slide the pebble on x'_i to x_i and finally shift our pebble from a_i to q_i. Once we place a pebble on node q_i we can discard pebbles from every other node in G_{x_i} so that the total number of pebbles on the graph is $1 + 3(n - i - 1)$ (3 pebbles on G_{x_j} for each $n \geq j > i$) and our invariant is maintained.

C.2 Red-Blue Pebbling Strategy

Setting our cache size $m = 3n + 3$ we would like to claim that G_ϕ also has higher red-blue pebbling cost whenever ϕ is not satisfiable. Intuitively, a black pebbling which only uses $3n + 3$ pebbles corresponds to a red-blue pebbling strategy with no expensive blue moves i.e., $3n+3$ red pebbles are sufficient. Unfortunately, the claim is not true about the graph G_ϕ. In particular, the optimal red-blue pebbling may not place each variable gadget G_{x_i} in a true or false configuration. In particular, instead of placing a variable gadget x_i in the false configuration x_i it would be better to maintain red-pebbles on nodes x_i and $\overline{x}_i$. Instead of discarding a pebble on node $\overline{x}'_i$ we simply place a blue pebble on this node. This allows us to avoid re-pebbling the pyramid Δ_{3i+1} later on when moving our pebble from node q_{i-1} to q_i. This strategy incurs two extra blue moves (cost: $2c_b$) but saves at least $\sum_{i=1}^{3i+1} i$ red moves (cost: $\Theta(i^2 c_r)$). We address the issue by adding an additional path gadget to form a new graph H_ϕ. Intuitively, the path gadget forces us to pebble every node in Δ_{3i+1} twice. We can then prove that H_ϕ has higher red-blue pebbling cost (with $m = 3n + 4$) whenever ϕ is not satisfiable. Intuitively, when ϕ is not satisfiable the pebbling will need to make at least 1 blue move without reducing the number of red-moves (each node in Δ_{3i+1} still needs to be pebbled twice). If ϕ is satisfiable a red-blue pebbling will essentially follow the same strategy for G_ϕ to avoid any blue moves with a few additional steps to pebble the path gadget.

Lemma C.1. [23] *The quantified Boolean formula*

$$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n F_n$$

is true if and only if the corresponding DAG G_{TQBF} has pebbling number $3n + 3$.

D NP-Hardness of the Red-Blue Pebbling Cost

In this section, we consider the computational complexity of computing $\mathsf{rbpeb}^{\|}(G, m)$, defining a decision version below and showing it is $\mathtt{NP-Hard}$.

The decision problem $\mathsf{rbpeb}^{\|}$ is defined as follows:
Input: a DAG G on n nodes, parameter c_b, c_r, and integers $m, d > 0$.
Output: *Yes*, if $\mathsf{rbpeb}^{\|}(G, m) \leq d$; otherwise *No*.

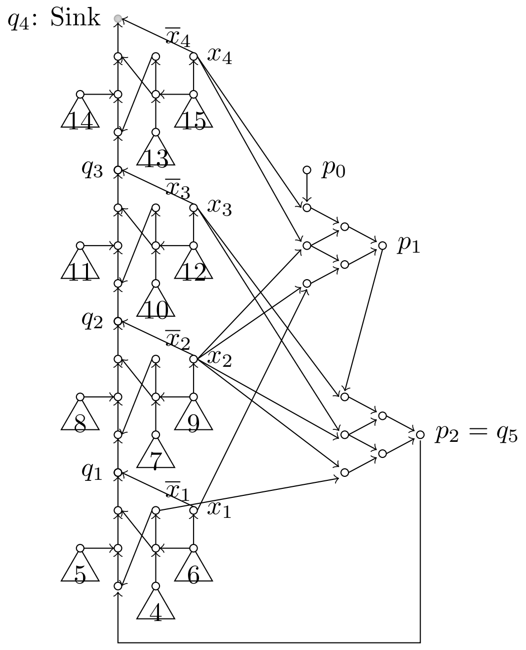

Fig. 4: Graph G_{TQBF} for $\exists x_1, x_2, x_3, x_4$ s.t. $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_1)$.

We now show that it is $\texttt{NP} - \texttt{Hard}$ to compute $\mathsf{rbpeb}^{||}(G, m)$. Quanquan Liu [26] observed that when $c_r = 0$ the problem is $\mathsf{PSPACE} - \mathsf{Hard}$ via a straightforward reduction from minimum space black pebbling. As we observed previously, when $c_b/c_r \in \mathcal{O}(\mathsf{poly}(n))$ the decision problem is in $\texttt{NP}$ and has a fundamentally different structure. We show that even when the cost of red moves is significant, the problem remains $\texttt{NP} - \texttt{Hard}$. We first reduce from a version of $3 - \mathsf{SAT}$ in which each variable appears in exactly 4 clauses and the negation of each variable also appears in exactly 4 clauses. Moreover, no consecutive $\frac{n}{105}$ clauses share the same variable (or negation). We show this version of $3 - \mathsf{SAT}$ is $\texttt{NP} - \texttt{Hard}$ in Theorem D.2, but first we show that even if each variable and negation appear in exactly 4 clauses, determining whether a 3CNF is satisfiable is $\texttt{NP} - \texttt{Hard}$.

Lemma D.1. *Let ϕ be a 3CNF formula with n variables and $m = \frac{8}{3}n$ clauses such that each variable appears in exactly 4 clauses and the negation of each variable also appears in exactly 4 clauses. Then determining whether ϕ is satisfiable is $\texttt{NP} - \texttt{Hard}$.*

Proof. [32] shows that if ϕ' is a 3CNF formula with n variables such that each variable or its negation appears in at most 4 clauses each and no clause contains the same literal multiple times, then determining whether ϕ' is satisfiable is $\texttt{NP} - \texttt{Hard}$. We show that ϕ' can be transformed into a 3CNF formula ϕ so that each variable and its negation appear in exactly 4 clauses each.

We first transform ϕ' so that each variable and its negation appear exactly 4 times. For each variable x_i that does not appear 4 times, we can force x_i to appear 4 times by appending ϕ' with the clause $(x_i \vee x_{n+j} \vee \neg x_{n+j})$ for a new variable x_{n+j} that has not previously appeared in ϕ'. We can do this until all n original variables and their negations appear exactly 4 times each. Now we may have some variables x_j, $\neg x_j$ for $j > n$ that only appear once. We thus append further ϕ' by additional clauses with variables x_k, x_{k+1}, x_{k+2} that have not appeared in ϕ', but are set to true. Namely, we append ϕ' with $(x_j \vee x_k \vee \neg x_k), (x_j \vee x_k \vee \neg x_k), (x_j \vee x_k \vee \neg x_k), (\neg x_j \vee x_k \vee \neg x_k), (\neg x_j \vee x_{k+1} \vee \neg x_{k+1}), (\neg x_j \vee x_{k+1} \vee \neg x_{k+1}), (x_{k+1} \vee x_{k+2} \vee \neg x_{k+2}), (x_{k+1} \vee x_{k+2} \vee \neg x_{k+2}), (\neg x_{k+1} \vee x_{k+2} \vee \neg x_{k+2}), (\neg x_{k+1} \vee x_{k+2} \vee \neg x_{k+2})$. Since we add at most $21n$ variables x_j with $j > n$, then the total number of variables in the resulting ϕ' is at most $22n$ and the total number of clauses is at most $64n$. Note that these extra clauses are inherently satisfiable, but do not affect the original clauses, so that resulting 3CNF formula is satisfiable if and only if

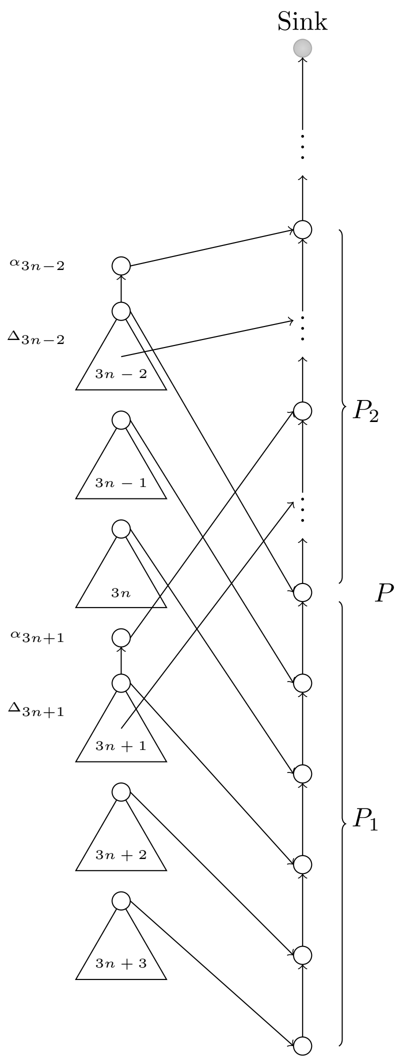

Fig. 5: Path P that is used in H_ϕ.

the original 3CNF formula is satisfiable. Since it is $\texttt{NP}-\texttt{Hard}$ to determine whether ϕ' is satisfiable, then it is also $\texttt{NP}-\texttt{Hard}$ to determine whether ϕ is satisfiable. □

We now show that such a 3CNF formula can be written so that no consecutive $\frac{n}{105}$ clauses share the same variable (or negation).

Theorem D.2. *Let ϕ be a 3CNF formula with n variables and $c = \frac{8}{3}n$ clauses such that each variable appears in exactly 4 clauses and the negation of each variable also appears in exactly 4 clauses. Furthermore, suppose that no consecutive $\frac{n}{105}$ clauses share the same variable (or negation). Then determining whether ϕ is satisfiable is* $\texttt{NP}-\texttt{Hard}$.

Proof. Let ϕ' be a 3CNF formula with n variables such that each variable or its negation appears in at most 4 clauses each and no clause contains the same literal multiple times. We now reorder ϕ' to obtain a 3CNF formula ϕ so that no consecutive $\frac{n}{105}$ clauses share the same variable. We use a greedy strategy to construct the first part of ϕ. We arbitrarily pick a clause in ϕ' to be the first clause of ϕ and then repeatedly append clauses in ϕ' that that do not share variables with any of the last $\frac{n}{105}$ clauses, until this is no longer possible.

Then there are at most $\frac{n}{35}$ variables in the last $\frac{n}{105}$ clauses, so there are at most $r \le \frac{(8-1)n}{35} = \frac{n}{5}$ remaining clauses in ϕ' that use one of these variables.

For each remaining clause c_i, we search for an interval of $\frac{n}{50}$ clauses that do not intersect with c_i and insert c_i in the middle of this interval. Such an interval must exist since there are at least 50 such disjoint intervals and each of the 3 variables appearing in c_i can intersect with at most 8 of these intervals. Thus ϕ has the desired form that each variable and its negation appear in exactly 4 clauses each and no consecutive $\frac{n}{105}$ clauses share the same variable (or negation). Moreover, ϕ is satisfiable if and only if ϕ' is satisfiable by construction. Since it is $\mathtt{NP}-\mathtt{Hard}$ to determine whether ϕ' is satisfiable, then it is also $\mathtt{NP}-\mathtt{Hard}$ to determine whether ϕ is satisfiable. $\square$

We now use a 3CNF formula satisfying the form of Theorem D.2 to show that the problem $\mathsf{rbpeb}^{\parallel}$ is $\mathtt{NP}-\mathtt{Hard}$.

Theorem D.3. *For $c_b > 10000c_r$, the problem* $\mathsf{rbpeb}^{\parallel}$ *is* $\mathtt{NP}-\mathtt{Hard}$.

Gilbert *et al.* showed that the minimum space black pebbling problem was $\mathsf{PSPACE}-\mathsf{Hard}$ by reduction from the Truly Quantified Boolean Formula (TQBF) problem. For more details about the Gilbert *et al.* [23] reduction, we refer an interested reader to Appendix C. We note that an instance ϕ of $3-\mathsf{SAT}$ with n variables and c clauses is still a TQBF instance (albeit with no $\forall$ quantifiers). Thus, given an instance ϕ of $3-\mathsf{SAT}$ satisfying the conditions in Theorem D.2 with n variables and c clauses, we can create the corresponding DAG G_ϕ, as described in the reduction of Gilbert *et al.* [23]. The graph G_ϕ has the property that it can be pebbled with at most $3n+3$ black pebbles if and only if ϕ is satisfiable.

In particular, the optimal pebbling for G_ϕ first uses $3n+3$ pebbles for Δ_{3n+3} and then leaves three pebbles on the corresponding existential quantifier for x_n, including a pebble at the node corresponding to the value of x_n, so that the sink node q_n can eventually be pebbled. The optimal pebbling then uses $3n$ additional pebbles for Δ_{3n} and determines the value of x_{n-1}, so that the total number of pebbles at any point is still at most $3n+3$. This process continues so that pebbling G_ϕ requires at least $3n+3$ pebbles until there is a value for each variable and the sink node can be pebbled. On the other hand, if ϕ is not satisfiable, then some variable must be "set" to both true and false, requiring an additional pebble. Hence the pebbling requires at least $3n+4$ black pebbles.

In fact, if variable x_i is "set" to both true and false, then the nodes in Δ_{3i+3} and the node $\overline{x}'_i$ need to be pebbled twice in a legal black pebbling. For red-blue pebblings, we could potentially use an extra blue move to store the sink of Δ_{3i+3} rather than completely repebbling Δ_{3i+3}. Thus we create an extra gadget that requires Δ_{3i+3} and $\overline{x}'_i$ to be completely repebbled, so that the strategy of storing $\overline{x}'_i$ and the sink of Δ_{3i+3} is useless.

We detail a gadget to append to G_ϕ to create a graph H_ϕ so that $\mathsf{rbpeb}^{\parallel}(H_\phi, m) = d := \frac{6n^3+27n^2+61n+20+12c}{2}$ if ϕ is a satisfiable assignment, but $\mathsf{rbpeb}^{\parallel}(H_\phi, m) > d$ if ϕ is not satisfiable. The key goal of the additional gadget is to ensure that we cannot *significantly* reduce the number of red moves (computation costs) by including a few blue moves. Moreover, by setting $m \ge 3n+4$ to be large, then there is no restriction on the number of red moves.

For DAG G_ϕ corresponding to n variables, there exist unique k-pyramids for $k = 4, \ldots, 3n+2, 3n+3$. Let Δ_i be the i-pyramid and let α_i be the vertex *above* the apex of pyramid Δ_i. Let P_1 be a directed path with $3n$ vertices so that there exists an edge from the apex of Δ_{3n+4-i} to vertex i of P_1, for each $1 \le i \le 3n$. Thus P_1 requires all sinks of the pyramids to be pebbled. See Figure 5 for an example of P_1.

We then connect the final vertex of P_1 to a directed path P_2 with

$$\left(\frac{(3n+2)(3n+1)}{2}+1\right)+\left(\frac{(3n-1)(3n-2)}{2}+1\right)+\ldots+(28+1)+(10+1)=\frac{3n^3+9n^2+10n}{2}$$

vertices. Moreover, the first $\frac{(3n+2)(3n+1)}{2}$ vertices of P_2 each have an edge from separate vertices of Δ_{3n+1}, starting with the vertices in the bottom layer and moving upwards. More specifically, let $u_1, \ldots, u_{3n+1}$ be the $3n+1$ vertices at the bottom layer of Δ_{3n+1} and $v_1, \ldots, v_{3n+1}$ be the first $3n+1$ vertices of P_2. Then there exist edges (u_i, v_i) for each $i \in [3n+1]$. Similarly, let $y_1, \ldots, y_{3n}$ be the $3n$ vertices at the next layer of

Δ_{3n+1} and $z_1, \ldots, z_{3n}$ be the next $3n$ vertices of P_2, following v_{3n+1}. Then there exist edges (y_i, z_i) for each $i \in [3n]$, and so forth until all vertices of Δ_{3n+1} have an outgoing edge to a separate vertex of P_2. We also create an edge to the following vertex from the vertex α_{3n+1}. This ensures that Δ_{3n+1} must be completely repebbled, so that any strategy of saving a pebble on a particular node of Δ_{3n+1}, such as its sink α_{3n+1}, is useless.

The next $\frac{(3n-1)(3n-2)}{2}$ vertices of P_2 each have an edge from separate vertices of Δ_{3n-2}, starting with the vertices in the bottom layer and moving upwards. We also create an edge to the following vertex from the vertex α_{3n-2}. We continue this process until all vertices from all pyramids of the form Δ_{3i+1} are connected to P_2, as well as the vertices α_{3i+1}. Finally, we connect P_2 to the same sink node as G_ϕ. Thus P_2 ensures that all pyramids of the form Δ_{3i+1} must be completely repebbled. See Figure 5 for an example of P_2.

Then by setting P to be the path P_1 concatenated with P_2, we have the following result:

Lemma D.4. *P contains exactly $3n + 3 + \sum_{i=1}^{n} \left(\frac{(3i+2)(3i+1)}{2} + 1\right) = \frac{3n^3+9n^2+16n+6}{2}$ vertices.*

Let $H_\phi = G_\phi \cup P$ and recall that P and G_ϕ have the same sink node. We claim that H_ϕ with capacity $3n + 4$ will have a certain pebbling cost if and only if ϕ is satisfiable. Thus, if ϕ is satisfiable, the optimal pebbling will correspond to the minimum space black pebbling and will require 0 blue moves. We first claim that if ϕ is unsatisfiable, then H_ϕ has pebbling number at least $3n + 5$.

Lemma D.5. *H_ϕ has pebbling number $3n + 4$ if and only if ϕ is satisfiable.*

Proof. We first note that if ϕ is satisfiable, then H_ϕ has pebbling number $3n + 3$. Recall that there exists a valid pebbling Q of G_ϕ with pebbling number $3n + 3$ that begins with all $3n + 3$ pebbles on the pyramid graph Δ_{3n+3} at some point. When the apex of Δ_{3n+3} is pebbled by Q, we can begin pebbling P in the next step. We keep a single pebble on path P and move the pebble forward along P_1 whenever the apex of the next pyramid is pebbled. The pebbling strategy Q must then pebble each of the pyramids $\Delta_{3n+2}, \ldots, \Delta_4$ in that order, which allows us to completely pebble the path P_1 using at most $3n+3$ pebbles in total. We then proceed with the pebbling strategy Q, observing that the sink of G_ϕ has two parents: a node representing the variable x_n set to true and some other node, say β. At some point Q will pebble β, at which point we maintain pebbles on β and P. We then hold the pebble on β while we pebble P_2, which can be done using $3n + 3$ additional pebbles. When the final node of P_2 is pebbled, we can use β to pebble the sink node of G_ϕ, using $3n + 4$ pebbles in total.

Suppose by way of contradiction, there exists an unsatisfiable ϕ such that each pebbling $Q = \{Q_1, Q_2, \ldots\}$ of H_ϕ has pebbling number at most $3n + 4$. By Lemma C.1, G_ϕ has pebbling number at least $3n + 4$ if ϕ is unsatisfiable. Thus there exists a time in which there are at least $3n+4$ pebbles on G_ϕ, i.e., $|Q_t \cap G_\phi| \geq 3n+4$. Let t be the final time in the pebbling Q, in which there are at least $3n + 4$ pebbles on G_ϕ. Moreover, we can assume without loss of generality that the sink node of G_ϕ is not pebbled at time t, since the pebbling will not need any other pebbles in future steps, as the pebbling can terminate after pebbling the sink node. Since G_ϕ and P only intersect at the sink node and Q_t already has at least $3n + 4$ nodes at G_ϕ and no pebble on the sink node, then either Q_t contains at least $3n + 5$ pebbles or P_t has no pebbles on P, i.e., either $|Q_t| \geq 3n+5$ or $G_\phi \cap P = \emptyset$. We have by assumption that $|Q_t| \leq 3n+4$, so it follows that there must be no pebbles on P.

To pebble the sink node of H_ϕ, we must completely pebble P after time t. Thus we must pebble a pyramid graph Δ_{3n+3} while holding a pebble on P, while requiring $3n+4$ pebbles with no other pebbles on G_ϕ. However, because t is the final time in which Q has $3n+4$ pebbles on G_ϕ, then Q can no longer pebble the sink node of G_ϕ, which is a contradiction. Hence, H_ϕ has pebbling number $3n + 4$ if and only if ϕ is satisfiable. $\square$

Lemma D.6. *If ϕ is satisfiable, then there exists a pebbling strategy of H_ϕ with capacity $3n+4$ and cost at most*

$$\left(\frac{6n^3 + 27n^2 + 101n + 20}{2}\right) c_r.$$

Proof. The total number of nodes in G_ϕ corresponding to variable assignments from the GLT construction is

$$6n + \sum_{i=4}^{3n+3} i = \frac{9n^2+33n+12}{2},$$

since each existential quantifier gadget has six internal nodes in addition to the pyramids of size $4, \ldots, 3n+3$. This can be visualized in Figure 4 by the nodes on the left hand side, excluding the nodes q_i. Additionally, there are n nodes q_i, six nodes for each of the c clauses p_i for $1 \le i \le c$, and an additional node for p_0. Moreover, it should be noted that since both x_i and $\overline{x_i}$ appear in 4 clauses, then regardless of the configuration in Figure 3, $4n$ additional pebbles are required for G_ϕ, either $4n$ pebbles on x_i or $4n$ pebbles on $\overline{x_i}$. Thus the total number of nodes that must be pebbled in G_ϕ is

$$4n + 6c + 1 + 7n + \sum_{i=4}^{3n+3} i = \frac{9n^2+35n+14}{2} + 6c + 4n = \frac{9n^2+75n+14}{2},$$

where the last equality results from the fact that $c = \frac{8}{3}n$.

By Lemma D.4, the number of nodes in the additional path P is $\frac{3n^3+9n^2+16n+6}{2}$. Moreover, we can completely re-pebble each of the pyramids Δ_{3i+1} a second time, as well as each α_{3i+1}, to pebble P, requiring an additional

$$\sum_{i=1}^{n} \left(\frac{(3i+2)(3i+1)}{2} + 1 \right) = \frac{3n^3+9n^2+10n}{2}$$

steps. Namely, we walk a pebble down P so that the pebble is placed on each node of P for a single step. Accordingly, we begin pebbling each pyramid so that its apex contains a pebble in the round before the descendent of the apex in P contains a pebble.

Thus, the total number of steps required to pebble H_ϕ is

$$\frac{9n^2+75n+14}{2} + \frac{3n^3+9n^2+16n+6}{2} + \frac{3n^3+9n^2+10n}{2} = \frac{6n^3+27n^2+101n+20}{2}.$$

Finally, recall from Lemma C.1 that the GLT construction has pebbling number $3n+3$ for a satisfiable instance of ϕ. Since the nodes in P are ordered corresponding to the natural pebbling order in G_ϕ, a single additional pebble suffices for P. Thus, if the capacity of G_ϕ is $3n+4$, then all pebbling moves can be achieved with red moves, so there exists a pebbling strategy with total cost is $\left(\frac{6n^3+27n^2+101n+20}{2}\right) c_r$. □

Lemma D.7. *If ϕ is unsatisfiable, then the pebbling cost of H_ϕ with capacity $3n+4$ is greater than*

$$\left(\frac{6n^3+27n^2+101n+20}{2}\right) c_r.$$

Proof. If ϕ is unsatisfiable, then H_ϕ has pebbling number at least $3n+5$ by Lemma D.5. Thus if H_ϕ has capacity $3n+4$, then H_ϕ any red-blue pebbling strategy must have a blue pebble at some point. Suppose that our pebbling strategy makes k blue moves e.g., by placing blue pebbles on the top of $3i+2$ pyramids. The only way such a strategy could be beneficial is if there is a large reduction in the number of red moves. We observe that in the pebbling strategy from Lemma D.6 almost all nodes are pebbled only once with the exception of (1) pyramids Δ_{3i+1}, which are each pebbled twice, and (2) the vertices corresponding x_i and/or $\overline{x_i}$.

This pebbling strategy incurs $4n$ extra red moves on vertices $\bigcup_{i \le n} \{x_i, \overline{x}_i\}$. We also remark that any pebbling strategy will need to place a red pebble on every node at least once. Since blue moves are more expensive the only reason to place a blue pebble on a node is if this allows us to reduce the number of red moves. Suppose that our pebbling strategy places k' blue pebbles on pyramids Δ_{3i+1} and k blue pebbles on other nodes. We claim that the total cost of the red pebbling moves can be reduced by at most $105(8/3)(k+2k'+3)(4c_r) + k'c_r$.

Suppose we place k' blue pebbles on pyramids Δ_{3i+1}. We can either keep blue pebbles on internal nodes of the pyramids or on top of the pyramid Δ_{3i+1}. Each blue pebble kept on some node of a pyramid Δ_{3i+1} can save an additional red move in the pebbling strategy, but it does not free up any room for additional red pebbles in cache because the honest pebbling strategy does not store red pebbles on this pyramid. Thus the total cost of the red moves saved by the k' blue pebbles is at most $k'c_b$.

Suppose we place k blue pebbles on nodes that are not in pyramids Δ_{3i+1}. Then each blue pebble will not save any red moves on the pyramids Δ_{3i+1}, but can save some of the $4n$ red moves on the nodes $\{x_i, \overline{x}_i\}$. For each $i \in [n]$, we define the indicator variable $Y_i = 1$ if and only if we reduce the red cost on variable gadget i to anything below $4c_r$. Observe that if $\sum Y_i > 105(8/3)(T)$, then there would be some point in time t where more than T variable gadgets have pebbles on both nodes x_i and $\overline{x}_i$. Suppose by way of contradiction that $T > k + 2k' + 3$. Then we would have at least 4 pebbles on $T - k'$ variable gadgets and at most k' variable gadgets with only two pebbles, for a total of $3n + (T - k') - k' + 2 \geq 3n + k + 5$ pebbles (extra two pebbles on path P and at least one on clause gadget). However, this contradicts the fact that we have at most $3n + 4 + k$ total pebbles (red and blue) at all times in the pebbling. Thus, we have $T \leq k + 2k' + 3$, so that $\sum Y_i \leq 105(8/3)(k + 2k' + 3)$ and the total cost of the red moves saved by the k blue pebbles is at most $105(8/3)(k + 2k' + 3)(4c_r)$.

In summary, for $k + k' \geq 1$, the total cost of blue moves is $(k + k')c_b$ and the total number of saved red moves is at most $105(8/3)(k+2k'+3)(4c_r) + k'c_r$. Thus for $c_b > 10000c_r$, we have $(k+k')c_b > 105(8/3)(k+2k'+3)(4c_r) + k'c_r$. Therefore, any pebbling strategy has a cost greater than $\left(\frac{6n^3+27n^2+101n+20}{2}\right) c_r$. $\square$

Together, Lemma D.6 and Lemma D.7 imply Theorem D.3.

Reminder of Theorem D.3. *For $c_b > 10000c_r$, the problem* $\mathsf{rbpeb}^{\|}$ *is* $\mathtt{NP-Hard}$.

Proof of Theorem D.3: First, we remark that given a DAG H_ϕ with some capacity m, as well as a complete pebbling strategy as the certificate, the certificate can be verified in polynomial time by checking the validity of each step in the pebbling strategy. Thus, the computation of $\mathsf{rbpeb}^{\|}(H_\phi)$ is in $\mathtt{NP}$.

We now reduce $\mathsf{3-SAT}$ to the computation of $\mathsf{rbpeb}^{\|}(H_\phi)$. Now, given an instance ϕ of $\mathsf{3-SAT}$ with n variables, we construct the above DAG H_ϕ. This procedure clearly takes polynomial time. Moreover, by Lemma D.6, if ϕ is satisfiable, then the optimal pebbling cost of H_ϕ with capacity $3n+4$ is exactly

$$\left(\frac{6n^3 + 27n^2 + 101n + 20}{2}\right) c_r.$$

On the other hand, by Lemma D.7, if ϕ is unsatisfiable, then the pebbling cost of H_ϕ with capacity $3n+4$ is greater than

$$\left(\frac{6n^3 + 27n^2 + 101n + 20}{2}\right) c_r.$$

Thus, the computation of $\mathsf{rbpeb}^{\|}(H_\phi, m)$ distinguishes whether ϕ is satisfiable or not, for $m \geq 3n+4$ and $d = \frac{6n^3+27n^2+101n+20}{2}$. Since $\mathsf{3-SAT}$ is $\mathtt{NP-Hard}$, it follows that the $\mathsf{rbpeb}^{\|}(H_\phi, m)$ is $\mathtt{NP-Hard}$. $\square$