Computationally Data-Independent Memory Hard Functions

Mohammad Hassan Ameri 🆔
Department of Computer Science, Purdue University, West Lafayette, IN, USA
https://www.cs.purdue.edu/homes/mameriek/
mameriek@purdue.edu

Jeremiah Blocki 🆔
Department of Computer Science, Purdue University, West Lafayette, IN, USA
https://www.cs.purdue.edu/homes/jblocki
jblocki@purdue.edu

Samson Zhou 🆔
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
https://samsonzhou.github.io/
samsonzhou@gmail.com

─── Abstract ───

Memory hard functions (MHFs) are an important cryptographic primitive that are used to design egalitarian proofs of work and in the construction of moderately expensive key-derivation functions resistant to brute-force attacks. Broadly speaking, MHFs can be divided into two categories: data-dependent memory hard functions (dMHFs) and data-independent memory hard functions (iMHFs). iMHFs are resistant to certain side-channel attacks as the memory access pattern induced by the honest evaluation algorithm is independent of the potentially sensitive input e.g., password. While dMHFs are potentially vulnerable to side-channel attacks (the induced memory access pattern might leak useful information to a brute-force attacker), they can achieve higher cumulative memory complexity (CMC) in comparison than an iMHF. In particular, any iMHF that can be evaluated in N steps on a sequential machine has CMC *at most* $\mathcal{O}\left(\frac{N^2 \log \log N}{\log N}\right)$. By contrast, the dMHF scrypt achieves maximal CMC $\Omega(N^2)$ – though the CMC of scrypt would be reduced to just $\mathcal{O}(N)$ after a side-channel attack.

In this paper, we introduce the notion of computationally data-independent memory hard functions (ciMHFs). Intuitively, we require that memory access pattern induced by the (randomized) ciMHF evaluation algorithm appears to be independent from the standpoint of a computationally bounded eavesdropping attacker – even if the attacker selects the initial input. We then ask whether it is possible to circumvent known upper bound for iMHFs and build a ciMHF with CMC $\Omega(N^2)$. Surprisingly, we answer the question in the affirmative when the ciMHF evaluation algorithm is executed on a two-tiered memory architecture (RAM/Cache).

We introduce the notion of a k-restricted dynamic graph to quantify the continuum between unrestricted dMHFs ($k = n$) and iMHFs ($k = 1$). For any $\epsilon > 0$ we show how to construct a k-restricted dynamic graph with $k = \Omega(N^{1-\epsilon})$ that provably achieves maximum cumulative pebbling cost $\Omega(N^2)$. We can use k-restricted dynamic graphs to build a ciMHF provided that cache is large enough to hold k hash outputs and the dynamic graph satisfies a certain property that we call "amenable to shuffling". In particular, we prove that the induced memory access pattern is indistinguishable to a polynomial time attacker who can monitor the locations of read/write requests to RAM, but not cache. We also show that when $k = o\left(N^{1/\log \log N}\right)$, then any k-restricted graph with constant indegree has cumulative pebbling cost $o(N^2)$. Our results almost completely characterize the spectrum of k-restricted dynamic graphs.

1 Introduction

Memory hard functions (MHFs) [1, 27] are a central component in the design of password hashing functions [9], egalitarian proofs of work [21], and moderately hard key-derivation functions [27]. In the setting of password hashing, the objective is to design a function that can be computed relatively quickly on standard hardware for honest users, but is prohibitively expensive for an offline attacker to compute millions or billions of times while checking each password in a large cracking dictionary. The first property allows legitimate users to authenticate in a reasonable amount of time, while the latter goal discourages brute-force offline guessing attacks, even on low-entropy secrets such as passwords, PINs, and biometrics. The objective is complicated by attackers that use specialized hardware such as Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs) to significantly decrease computation costs by several orders of magnitude, compared to an honest user using standard hardware. For example, the Antminer S17, an ASIC Bitcoin miner exclusively configured for SHA256 hashes, can compute up to 56 trillion hashes per second, while the rate of many standard CPUs and GPUs are limited to 200 million hashes per second and 1 billion hashes per second, respectively.

Memory hard functions were developed on the observation that memory costs such as chip area tend to be equitable across different architectures. Therefore, the cost of evaluating an ideal "egalitarian" function would be dominated by memory costs. Blocki et al. [12] argued that key derivation functions without some form of memory hardness provide insufficient defense against a economically motivated offline attacker under the constraint of reasonable authentication times for honest users. In fact, most finalists in the 2015 Password Hashing Competition claimed some form of memory hardness [22, 9, 24]. To quantify these memory costs, memory hardness [27] considers the cost to build, obtain, and empower the necessary hardware to compute the function. One particular metric heavily considered by recent cryptanalysis [2, 5, 4, 3, 15] is cumulative memory complexity (CMC) [7], which measures the amortized cost of any parallel algorithm evaluating the function on several distinct inputs. Despite known hardness results for quantifying [16] or even approximating [13] a function's CMC, even acquiring asymptotic bounds provide automatic bounds for other attractive metrics such as space-time complexity [26] or energy complexity [29, 14].

Data-Dependent vs. Data-Independent Memory Hard Functions

At a high level, memory hard functions can be categorized into two design paradigms: data-dependent memory hard functions (dMHFs) and data-independent memory hard functions (iMHFs). dMHFs induce memory access patterns that depend on the input, but can achieve

high memory hardness with potentially relatively easy constructions [6]. However, dMHFs are also vulnerable to side-channel attacks due to their inherent data dependent memory access patterns [8]. Examples of dMHFs include scrypt [27], Argon2d [10] and Boyen's halting puzzles [19]. On the other hand, iMHFs have memory access patterns that are independent of the input, and therefore resist certain side-channel attacks such as cache timing [8]. Examples of iMHFs include 2015 Password Hashing Competition (PHC) winner Argon2i [9], Balloon Hashing [17] and DRSample [4]. iMHFs with high memory hardness can be more technically challenging to design, but even more concerning is the inability of iMHFs to be maximally memory hard.

Alwen and Blocki [2] proved that the CMC of any iMHF running in time N is at most $\mathcal{O}\left(\frac{N^2 \log \log N}{\log N}\right)$, while the dMHF scrypt has cumulative memory complexity $\Omega(N^2)$ [6], which matches the maximal amount and cannot be obtained by any iMHF. However, the cumulative memory complexity of a dMHF can be greatly decreased through a side-channel attack, if an attacker has learned the memory access pattern induced by the true input. Namely, a brute-force attacker can preemptively quit evaluation on a guess y once it is clear that the induced memory access pattern on input y differs from that on the true input x. For example, the cumulative memory complexity of scrypt after a side-channel attack is just $\mathcal{O}(N)$.

Ideally, we would like to obtain a family of memory hard functions with cumulative memory complexity $\Omega(N^2)$ without any vulnerability to side-channel attacks. A natural approach would be some sort of hybrid between data-dependent and data-independent modes, such as Argon2id, which runs the MHF in data-independent mode for $\frac{N}{2}$ steps before switching to data-dependent mode for the final $\frac{N}{2}$ steps. Although the cumulative memory complexity is the maximal $\Omega(N^2)$ if there is no side-channel attack, the security still reduces to that of the underlying iMHF (e.g., Argon2i) if there is a side-channel attack. Hence even for a hybrid mode, the cumulative memory complexity is just $\mathcal{O}\left(\frac{N^2 \log \log N}{\log N}\right)$ (or lower) in the face of a side-channel attack. Thus in this paper we ask:

> In the presence of side-channel attacks, does there exist a family of functions with $\Omega(N^2)$ cumulative memory complexity?

1.1 Our Contributions

Surprisingly, we answer the above question in the affirmative for a natural class of side-channel attacks that observe the read/write memory locations. We introduce the concept of computationally data-independent memory hard functions to overcome the inability of data-independent memory hard functions to be maximally memory hard [2] without the common side-channel vulnerabilities of data-dependent memory hard functions [8]. Our constructions work by randomly "shuffling" memory blocks in cache before they are stored in RAM (where the attacker can observe the locations of read/write requests). Intuitively, each time MHF.Eval(x) is executed the induced memory access pattern will appear different due to this scrambling step. The goal is to ensure that an attacker can not even distinguish between the observed memory access pattern on two known inputs $x \neq y$.

Towards this goal we define k-restricted dynamic graphs as a tool to quantify the continuum between dMHFs and iMHFs. Intuitively, in a k-restricted dynamic graph $G = (V = [N], E)$ we have parents(v) = $\{v - 1, r(v)\}$ where the second (data-dependent) parent $r(v) \in R_v$ must be selected from a fixed (data-independent) restricted set $R_v \subseteq V$ of size $|R_v| \leq k$. When $k = 1$ the function is an iMHF (the parent $r(v) \in R_v$ of each node v is fixed in a data-independent manner) and when $k = N$ the function is an unrestricted dMHF – scrypt

and Argon2d are both examples of unrestricted dMHFs. Intuitively, when k is small it becomes easier to scramble the labels R_v in memory so that the observed memory access patterns on two known inputs $x \neq y$ are computationally indistinguishable.

We then develop a graph gadget that generates a family of ciMHFs using k-restricted graphs. Using this family of ciMHFs, we characterize the tradeoffs between the value of k and the overall cumulative memory cost of k-restricted graphs.

Impossibility Results for Small k

Since k-restricted graphs correspond to iMHFs for $k = 1$, and it is known that $cc(\mathbb{G}) = \mathcal{O}\left(\frac{N^2 \log \log N}{\log N}\right)$ for any family $\mathbb{G}$ of iMHFs [2], then one might expect that it is impossible to obtain maximally memory hard ciMHFs for small k. Indeed, our first result shows that this intuition is correct; we show that for any $k = o\left(N^{1/\log \log N}\right)$, then any family of k-restricted graphs $\mathbb{G}$ with constant indegree has $cc(\mathbb{G}) = o(N^2)$.

▶ **Theorem 1.** *Let $\mathbb{G}$ be any family of k-restricted dynamic graphs with constant $indeg(\mathbb{G})$. Then*

$$cc(\mathbb{G}) = \mathcal{O}\left(\frac{N^2}{\log \log N} + N^{2 - 1/2 \log \log N}\sqrt{k^{1 - 1/\log \log N}}\right).$$

Thus for $k = o\left(N^{1/\log \log N}\right)$, we have $cc(\mathbb{G}) = o(N^2)$.

We prove this result in Theorem 10 and Corollary 11 in Section 3 by generalizing ideas from the pebbling attack of Alwen and Blocki [2] against any iMHF to k-restricted dynamic graphs graphs. The pebbling attack of Alwen and Blocki [2] exploited the fact that any constant indegree DAG G is somewhat depth-reducible e.g., we can always find a set $S \subseteq V(G)$ of size $e = \mathcal{O}\left(\frac{N \log \log N}{\log N}\right)$ such that any path in $G - S$ has length at most $d = \frac{N}{\log^2 N}$. The attack then proceeds in a number of *light phases* and *balloon phases*, where the goal of light phase i is to place pebbles on the interval $[ig + 1, (i + 1)g]$, for some parameter g to be optimized. At the same time, the attacker discards pebbles on all nodes v unless $v \in S$ or unless v is a parent of one of the next g nodes $[ig + 1, (i + 1)g]$ that we want to pebble. Once light phase i is completed, balloon phase i uses the pebbles on S to recover all previously discarded pebbles. Note that balloon phase i thus promises that pebbles are placed on the parents of the nodes $[(i + 1)g + 1, (i + 2)g]$, so that light phase $i + 1$ can then be initiated and so forth.

One key difference is that we must maintain pebbles on all gk nodes $u \in \bigcup_{v \in [ig+1,(i+1)g]} R_v$ that are "potential parents" of the next g nodes $[ig+1, (i+1)g]$. The total cost of the pebbling attack is $\mathcal{O}\left(eN + gkN + \frac{N^2 d}{g}\right)$, which is identical to [2] when $k = 1$ for (e, d)-reducible DAGs. In general for small values of k, the dynamic pebbling strategy can still achieve cumulative memory cost $o(N^2)$ after optimizing for g.

Maximally Hard k-restricted dMHF

In Section 4, we show how to construct a k-restricted dynamic graph for $k = \mathcal{O}\left(N^{1-\epsilon}\right)$, which has cumulative pebbling cost $\Omega(N^2)$ for any constant $\epsilon > 0$. Intuitively, our goal is to force the pebbling strategy to maintain $\Omega(N)$ pebbles on the graph for $\Omega(N)$ steps or pay a steep penalty. In particular, we want to ensure that if there are $o(N)$ pebbles on the graph at time i then the cumulative pebbling cost to advance a pebble just $2k = \mathcal{O}(N^\epsilon)$ steps is at least $\Omega(N^{2-\epsilon})$ with high probability. This would imply that the pebbling strategy either keeps $\Omega(N)$ pebbles on the graph for $\Omega(N)$ steps or that the pebbling strategy pays a penalty of $\Omega(N^{2-\epsilon})$ at least $\Omega\left(\frac{N}{k}\right) = \Omega(N^\epsilon)$ times. In either case the cumulative pebbling cost will be $\Omega(N^2)$.

One of our building blocks is the "grates" construction of Schnitger [30] who, for any $\epsilon > 0$, showed how to construct a constant indegree DAG G_ϵ that is (e, d)-depth robust graph with $e = \Omega(N)$ and $d = \Omega(N^{1-\epsilon})$. Our second building block is the superconcentrator [28, 26] graph. By overlaying the DAG G_ϵ with a superconcentrator, we can spread out the data-dependent edges on the top layer of our graph to ensure that (with high probability) advancing a pebble $2k = \mathcal{O}(N^\epsilon)$ steps on the top layer starting from a pebbling configuration with $o(N)$ pebbles on the graph requires us to repebble an (e, d)-depth robust graph with $e = \Omega(N)$ and $d = \Omega(N^{1-\epsilon})$. This is sufficient since Alwen et al. [5] showed that the cumulative pebbling cost of any (e, d)-depth robust graph is at least ed.

Open Question

We emphasize that we only show that any dynamic pebbling strategy for our k-restricted dynamic graph has cumulative cost $\Omega(N^2)$. This is not quite the same as showing that our dMHF has CMC $\Omega(N^2)$ in the parallel random oracle model. For static graphs, we know that the CMC of an iMHF is captured by the cumulative pebbling cost of the underlying DAG [7]. We take the dynamic pebbling lower bound as compelling evidence that the corresponding MHF has maximum cumulative memory cost. Nevertheless, proving (or disproving) that the CMC of a dMHF is captured by the cumulative cost of the optimal dynamic pebbling strategy for the underlying dynamic graph is still an open question that is outside the scope of the current work.

ciMHF Implementation Through Shuffling

The only problem is that the above k-restricted dynamic graph is actually a data-dependent construction; once the input x is fixed, the memory access patterns of the above construction is completely deterministic! Thus a side-channel attacker that obtains a memory access pattern will possibly be able to distinguish between future inputs. Our solution is to have a hidden random key K for each separate evaluation of the password hash. The hidden random key K does not alter the hash value of x in any manner, so we emphasize that there is no need to know the value of the hidden key K to perform computation. However, each computation using a separate value of K induces a different memory access pattern, so that no information is revealed to side-channel attackers looking at locations of read/write instructions.

Let L be a set of the last N consecutive nodes from our previous graph construction, which we suppose is called G_0. We form G by appending a path of length N to the end of G_0. We introduce a gadget that partitions the nodes in L into blocks $B_1, B_2, \ldots, B_{N/k}$ of size k each. We then enforce that for $i \in [N]$ and $j = i \mod k$, the ith node in the final N nodes of G has a parent selected uniformly at random from B_{j+1}, depending on the input x. Thus to compute the label of i, the evaluation algorithm should know the labels of all nodes in B_{j+1}.

We allow the evaluation algorithm to manipulate the locations of these labels so that the output of the algorithm remains the same, but each computation induces a different memory access pattern. Specifically, the random key K induces a shuffling of the locations of the information within each block of the block partition gadget. Thus if the size of each block is sufficiently large, then with high probability, two separate computations of the hash for the same password will yield distinct memory access patterns, effectively computationally data-independent. Then informally, a side-channel attacker will not be able to use the memory access patterns to distinguish between future inputs.

In fact, this approach works for a general class of graphs satisfying a property that we call "amenable to shuffling". We characterize the properties of the dynamic graphs that are amenable to shuffling in Section 5 and show that k-restricted dynamic graphs that are ameanble to shuffling can be used in the design of MHFs to yield computationally data-independent sequential evaluation algorithms.

▶ **Theorem 2.** *For each DAG G that is amenable to shuffling, there exists a computationally data-independent sequential evaluation algorithm computing a MHF based on the graph G that runs in time $\mathcal{O}(N)$. (Informal, see Theorem 24.)*

We believe that our techniques for converting graphs that are amenable to shuffling to ciMHFs may be of independent interest.

Finally, we provide a version of our dMHF with $\Omega(N^2)$ cumulative memory complexity that is amenable to shuffling. Combining this maximally hard k-restricted dMHF using a DAG that is amenable to shuffling with Theorem 2, we obtain a maximally hard ciMHF.

▶ **Theorem 3.** *Let $0 < \epsilon < 1$ be a constant and $k = \Omega(N^\epsilon)$. Then there exists a family $\mathbb{G}$ of k-restricted graphs with $\mathsf{cc}(\mathbb{G}) = \Omega(N^2)$ that is amenable to shuffling.*

We prove Theorem 3 in Section 6, introducing the necessary formalities for computationally data-independent memory hard functions and the underlying systems model. Our results in Theorem 1 and Theorem 3 almost completely characterize the spectrum of k-restricted graphs. In fact, for a graph G drawn uniformly at random from our distribution $\mathbb{G}$ in Theorem 3 and any pebbling strategy S, not only do we have $\underset{G \sim \mathbb{G}}{\mathbb{E}} [\mathsf{cc}(S, G)] = \Omega(N^2)$, but we also have $\mathsf{cc}(S, G) = \Omega(N^2)$ with high probability.

2   Preliminaries

We use the notation $[N]$ to denote the set $\{0, 1, \ldots, N-1\}$. For two numbers x and y, we use $x \circ y$ to denote their concatenation.

Given a directed acyclic graph (DAG) $G = (V, E)$ and a node $v \in V$, we use $\mathsf{parents}_G(v) = \{u \ : \ (u, v) \in E\}$ to denote the parents of node v. We use $\mathsf{ancestors}_G(v) = \bigcup_{i \geq 1} \mathsf{parents}_G^i(v)$ to denote the set of all ancestors of v – here, $\mathsf{parents}_G^2(v) = \mathsf{parents}_G(\mathsf{parents}_G(v))$ and $\mathsf{parents}_G^{i+1}(v) = \mathsf{parents}_G(\mathsf{parents}_G^i(v))$. We use $\mathsf{indeg}(v) = |\mathsf{parents}(v)|$ to denote the number of incoming edges into v and define $\mathsf{indeg}(G) = \underset{v \in V}{\max} \mathsf{indeg}(v)$. Given a set $S \subseteq V$, we use $G - S$ to refer to the graph obtained by deleting all nodes in S and all edges incident to S. We use $\mathsf{depth}(G)$ to denote the number of nodes in the longest directed path in G.

▶ **Definition 4.** *A DAG $G = (V, E)$ is (e, d)-reducible if there exists a subset $S \subseteq V$ of size $|S| \leq e$ such that any directed path P in G of length d contains at least one node in S. We call such a set S a* depth-reducing set. *If G is not (e, d)-reducible, then we say that G is (e, d)-depth robust.*

For a DAG $G = (V = [N], E)$, we use $G_{\leq i}$ to denote the subgraph of G induced by $[i]$. In other words, $G_{\leq i} = (V', E')$ for $V' = [i]$ and $E' = \{(a, b) \in E \mid a, b \leq i\}$.

The Parallel Random Oracle Model

We review the parallel random oracle model (pROM), as introduced by Alwen and Serbinenko [7]. There exists a probabilistic algorithm $\mathcal{A}^{\mathcal{H}}$ that serves as the main computational unit, where $\mathcal{A}^{\mathcal{H}}$ has access to an arbitrary number of parallel copies of an oracle $\mathcal{H}$ sampled uniformly at random from an oracle set $\mathbb{H}$ and proceeds to do computation in a number of

rounds. In each round i, $\mathcal{A}^{\mathcal{H}}$ maintains a state σ_i along with initial input x. $\mathcal{A}^{\mathcal{H}}$ determines a batch of queries $\mathbf{q}_i$ to send to $\mathcal{H}$, receives and processes the responses to determine an updated state σ_{i+1}. At some point, $\mathcal{A}^{\mathcal{H}}$ completes its computation and outputs the value $\mathcal{A}^{\mathcal{H}}(x)$.

We say that $\mathcal{A}^{\mathcal{H}}$ computes a function $f_{\mathcal{H}}$ on input x with probability ϵ if $\mathbf{Pr}\left[\mathcal{A}^{\mathcal{H}}(x) = f_{\mathcal{H}}\right] \geq \epsilon$, where the probability is taken over the internal randomness of $\mathcal{A}$. We say that $\mathcal{A}^{\mathcal{H}}$ uses t running time if it outputs $\mathcal{A}^{\mathcal{H}}(x)$ after round t. In that case, we also say $\mathcal{A}^{\mathcal{H}}$ uses space $\sum_{i=1}^{t} |\sigma_i|$ and that $\mathcal{A}^{\mathcal{H}}$ makes q queries if $\sum_{i=1}^{t} \leq q$.

The Ideal Cipher Model

In the ideal cipher model (ICM), there is a publicly available random block cipher, which has a κ-bit key K and an N bit input and output. Equivalently, all parties, including any honest parties and adversaries, have access to a family of 2^{κ} independent random permutations of $[N]$. Moreover for any given key K and $x \in [N]$, both encryption $\mathsf{Enc}(K, x)$ and decryption $\mathsf{Dec}(K, x)$ queries can be made to the random block cipher.

Graph Pebbling

The goal of the (black) pebbling game is to place pebbles on all sink nodes of some input directed acyclic graph (DAG) $G = (V, E)$. The game proceeds in rounds, and each round i consists of a number of pebbles $P_i \subseteq V$ placed on a subset of the vertices. Initially, the graph is unpebbled, $P_0 = \emptyset$, and in each round $i \geq 1$, we may place a pebble on $v \in P_i$ if either all parents of v contained pebbles in the previous round ($\mathsf{parents}(v) \subseteq P_{i-1}$) or if v already contained a pebble in the previous round ($v \in P_{i-1}$). In the sequential pebbling game, at most one new pebble can be placed on the graph in any round (i.e., $|P_i \backslash P_{i-1}| \leq 1$), but this restriction does not apply in the parallel pebbling game.

We use $\mathcal{P}_G^{\|}$ to denote the set of all valid parallel pebblings of a fixed graph G. The *cumulative cost* of a pebbling $P = (P_1, \ldots, P_t) \in \mathcal{P}_G^{\|}$ is the quantity $\mathsf{cc}(P) := |P_1| + \ldots + |P_t|$ that represents the sum of the number of pebbles on the graph during every round. The (parallel) *cumulative pebbling cost* of the fixed graph G, denoted $\mathsf{cc}(G) := \min_{P \in \mathcal{P}_G^{\|}} \mathsf{cc}(P)$, is the cumulative cost of the best legal pebbling of G.

▶ **Definition 5** (Dynamic/Static Pebbling Graph). *We define a* dynamic pebbling graph *as a distribution $\mathbb{G}$ over directed acyclic graphs $G = (V = [N], E)$ with edges $E = \{(i - 1, i) : i \leq N\} \cup \{(r(i), i) : i \leq N\}$, where $r(i) < i - 1$ is a randomly chosen directed edge. We say that an edge $(r(i), i)$ is* dynamic *if $r(i)$ is not chosen until a black pebbled is place on node $i - 1$. We say that the graph is* static *if none of the edges are dynamic.*

We now define a labeling of a graph G.

▶ **Definition 6.** *Given a DAG $G = (V = [N], E)$ and a random oracle function $H : \Sigma^* \to \Sigma^w$ over an alphabet Σ, we define the labeling of graph G as $L_{G,H} : \Sigma^* \to \Sigma^*$. In particular, given an input x the (H, x) labeling of G is defined recursively by*

$$L_{G,H,x}(v) = \begin{cases} H(v \circ x), & \mathsf{indeg}(v) = 0 \\ H\left(v \circ L_{G,H,x}(v_1) \circ \cdots \circ L_{G,H,x}(v_d)\right), & \mathsf{indeg}(v) > 0, \end{cases}$$

where $v_1, \ldots, v_d$ are the parents of v in G, according to some predetermined lexicographical order. We define $f_{G,H}(x) = L_{G,H,x}(s_1) \circ \ldots \circ L_{G,H,x}(s_k)$, where $s_1, \ldots, s_k$ are the sinks of G sorted lexicographically by node index. If there is a single sink node s_G then $f_{G,H}(x) = L_{G,H,x}(s_G)$. We omit the subscripts G, H, x when the dependency on the graph G and hash function H is clear. For a distribution of dynamic graphs $\mathbb{G}$, we say $f_{\mathbb{G},H}(x) = f_{G,H}(x)$ once a dynamic graph G has been determined from the choice of H and x.

For a node i, we define $\mathsf{PotentialParents}(i)$ to be set Y_i of minimal size such that $\mathbf{Pr}\left[r(i) \in Y_i\right] = 1$. We now define k-restricted dynamic graphs, which can characterize both dMHFs and iMHFs.

▶ **Definition 7** (k-Restricted Dynamic Graph). *We say that a dynamic pebbling graph $\mathbb{G}$ is k-restricted if for all i, $\mathsf{PotentialParents}(i) \leq k$.*

Observe that $k = 1$ corresponds to an iMHF while $k = N$ corresponds to a dMHF. Hence, k-restricted dynamic graphs can be viewed as spectrum between dMHFs and iMHFs.

We define the cumulative cost of pebbling a dynamic graph similar to the definition of cumulative cost of pebblings on static graphs. We first require the following definition of a dynamic pebbling strategy:

▶ **Definition 8** (Dynamic Pebbling Strategy). *A dynamic pebbling strategy S is a function that takes as input*
(1) *an integer $i \leq N$*
(2) *an initial pebbling configuration $P_0^i \subseteq [i]$ with $i \in P_0^i$*
(3) *a partial graph $G_{\leq i+1}$*
The output of $S(i, P_0^i, G_{\leq i+1})$ is a legal sequence of pebbling moves $P_1^i, \ldots, P_{r_i}^i$ that will be used in the next phase, to place a pebble on node $i+1$, so that $i+1 \in P_{r_i}^i \subseteq [i+1]$. Given $G \sim \mathbb{G}$ we can abuse notation and write $S(G)$ for the valid pebbling produced by S on the graph G i.e., $P_1^0, \ldots, P_{r_0}^0, P_1^1, \ldots, P_{r_1}^1, \ldots, P_{r_1}^{N-1}, \ldots, P_{r_{N-1}}^{N-1}$. Here, $P_1^i, \ldots, P_{r_i}^i = S(i, P_0^i, G_{\leq i+1})$ where $P_0^i = P_{r_{i-1}}^{i-1}$ and for $i = 1$ we set $P_0^0 = \emptyset$.

We thus define $\mathsf{cc}(S, G)$ to the pebbling cost of strategy S when we sample a dynamic graph G and $\mathsf{cc}(S, \mathbb{G}) = \mathop{\mathbb{E}}\limits_{G \sim \mathbb{G}} \left[\mathsf{cc}(S, G)\right]$. Finally, we define $\mathsf{cc}(\mathbb{G}) = \min_S \mathsf{cc}(S, \mathbb{G})$, where the minimum is taken over all dynamic pebbling strategies S. More generally, we define $\mathsf{cc}(S, \mathbb{G}, \delta) = \max\{k : \mathop{\mathbf{Pr}}\limits_{G \in \mathbb{G}} \left[\mathsf{cc}(S, G) \geq k\right] \geq 1 - \delta\}$. Fixing δ to be some negligible function of N, we can define $\mathsf{cc}_\delta(\mathbb{G}) = \min_S \mathsf{cc}(S, \mathbb{G}, \delta)$.

3 General Attack Against k-Restricted Graphs

In this section, we describe a general attack against k-restricted graphs. We show that the attack incurs cost $o(N^2)$ for $k = o\left(N^{1/\log\log N}\right)$, proving that there is no maximally memory hard k-restricted graph for small k.

We first require the following formulation of Valiant's Lemma, which shows the existence of a subroutine $\mathsf{Valiant}(G, e, d)$ to find a depth-reducing set S of size at most e within a graph G, for $e = \frac{\eta \delta N}{\log(N) - \eta}$ and $d = \frac{N}{2^\eta}$, where $\eta > 0$.

▶ **Lemma 9** (Valiant's Lemma). *[32] For any DAG $G = (V, E)$ with N nodes, indegree δ, and $\eta > 0$, there exists an efficient algorithm $\mathsf{Valiant}(G, e, d)$ to compute a set S of size $|S| \leq e := \frac{\eta \delta N}{\log(N) - \eta}$ such that $\mathsf{depth}(G - S) \leq d := \frac{N}{2^\eta}$.*

The high level intuition of the generic attack is as follows. By Valiant's Lemma (Lemma 9), $G \sim \mathbb{G}$ is (e, d)-reducible for $e = \frac{\eta \delta N}{\log(N) - \eta}$ and $d = \frac{N}{2^\eta}$. We will construct a dynamic pebbling strategy $\mathcal{A}$ that for all times t, maintains a depth-reducing set S_t such that $\mathsf{depth}(G_t - S_t) \leq d$, where G_t is the portion of G revealed after running $\mathcal{A}$ for time t, $G_t = G_{\leq i}$ for $i = 1 + \max \bigcup_{j=1}^t P_j$, where each $P_j \subseteq [N]$ represents the set of pebbled nodes during round j. Observe that for any i, $G_{\leq i}$ is (e, d)-reducible and hence G_t is also (e, d)-reducible for all times t. Thus, the depth reducing set S_t has size at most e for all times t and can be computed by a subroutine $\mathsf{Valiant}$, by Lemma 9. We now describe how $\mathcal{A}$ maintains this depth-reducing set through a series of *light phases* and *balloon phases*.

We first set a parameter g that we will eventually optimize. The goal of each light phase i is to pebble the next g nodes that have yet to be revealed. That is, if x_i is the largest node for which $\mathcal{A}$ has placed a pebble at some point prior to light phase i, then the goal of light phase i is to pebble the interval $[x_i + 1, x_i + g]$. To begin light phase i at some time t_i, we require that $(\mathsf{PotentialParents}([x_i + 1, x_i + g]) \cup S_{t_i}) \subseteq P_{t_i}$ for some depth-reducing set S_{t_i} of size at most e, such that $\mathsf{depth}(G_{t_i} - S_{t_i}) \le d$. Once this pre-condition is met, then light phase i simply takes g steps to pebble $[x_i + 1, x_i + g]$, since pebbles are already placed on $\mathsf{PotentialParents}([x_i + 1, x_i + g])$. Hence, the post-condition of light phase i at some time u_i is pebbles on the node $x_i + g$ and some depth-reducing set S_{u_i} of size at most e, such that $\mathsf{depth}(G_{u_i} - S_{u_i}) \le d$.

The goal of each balloon phase i is to place pebbles on all revealed nodes of the graph, to meet the pre-condition of light phase $i + 1$. To begin balloon phase i at some time r_i, we first have a necessary pre-condition that pebbles are placed on some depth-reducing set S_{r_i} of size at most e such that $\mathsf{depth}(G_{r_i} - S_{r_i}) \le d$. Once this pre-condition is met, then balloon phase i simply takes d steps to pebble the entire graph G_{r_i}, meeting the post-condition of balloon phase i.

We now formally prove the cumulative memory complexity of the attack in Algorithm 1.

▶ **Theorem 10.** *Let $\mathbb{G}$ be any family of k-restricted dynamic graphs with constant $\mathsf{indeg}(\mathbb{G})$. Then*

$$\mathsf{cc}(\mathbb{G}) = \mathcal{O}\left(\frac{N^2}{\log\log N} + N^{2 - 1/2 \log\log N} \sqrt{k^{1 - 1/\log\log N}} \right).$$

Proof. We analyze the cost of the pebbling strategy of Algorithm 1. Since G is drawn from a distribution of k-restricted dynamic graphs, then for any node x_i, $r(x_i)$ must be one of at most k labels. Thus for any consecutive g nodes, $|\mathsf{PotentialParents}([x_i + 1, x_i + g])| \le gk$. Hence it suffices to keep gk pebbles on the set of potential parents $\mathsf{PotentialParents}([x_i + 1, x_i + g])$ to pebble the interval $[x_i + 1, x_i + g]$, as well as a depth-reducing set of size at most e, for each of the g steps during light phase i. On the other hand, balloon phase i takes d steps, each of which trivially contains at most N pebbles.

$\mathcal{A}$ proceeds using $\frac{N}{g}$ total rounds of light and balloon phases, by pebbling g consecutive nodes at a time. Therefore, the total cost of the attack is $\mathcal{O}\left(Ngk + Ne + \frac{N}{g} \cdot dN \right)$, where the first and second terms originate from the cost of the light phases and the third term results from the cost of the balloon phases. Since we set $e = \frac{\eta \delta N}{\log(N) - \eta}$ and $d = \frac{N}{2^\eta}$ from Valiant's Lemma (Lemma 9) so that the total cost is $\mathcal{O}\left(\frac{\eta \delta N^2}{\log(N) - \eta} + Ngk + \frac{N^3}{2^\eta g} \right)$. By setting $g = \frac{N}{\sqrt{k 2^\eta}}$, the total cost is $\mathcal{O}\left(\frac{N^2 \eta \delta}{\log N - \eta} + N^2 \sqrt{\frac{k}{2^\eta}} \right)$. Finally, by setting $\eta = \frac{\log k + \log N - \log \delta}{\log\log N}$, the total cost is $\mathcal{O}\left(\frac{N^2}{\log\log N} + N^{2 - 1/2 \log\log N} \sqrt{k^{1 - 1/\log\log N}} \right)$.   ◀

Note that if $k = o(N^{1/\log\log N})$, then $\mathsf{cc}(\mathbb{G}) = o(N^2)$.

▶ **Corollary 11.** *Let $\mathbb{G}$ be any k-restricted dynamic graph with $k = o(N^{1/\log\log N})$ and constant indegree. Then $\mathsf{cc}(\mathbb{G}) = o(N^2)$.*

Algorithm 1 Generic pebbling strategy against dynamic DAG G.

Input: An integer i, an initial pebbling configuration $P_0^i \subseteq [i]$ with $i \in P_0^i$, a partial graph
    $G_{\leq i+1}$, and parameters d, e, g.
Output: A legal pebbling of $G_{\leq i+1}$.
 1: invariant ← True
 2: **if** $i \pmod{g} \equiv 0$ and $\mathsf{depth}(G_{\leq i+1} - P_0^i) > d$ **then**
 3:     invariant ← False
 4: **else if** $\mathsf{depth}(G_{\leq i+1} - P_0^i) > d$ or $\{i\} \cup \mathsf{PotentialParents}([i+1, i+g]) \nsubseteq P_0^i$ **then**
 5:     invariant ← False
 6: **if** invariant **then**   ▷ If pre-conditions met.
 7:     **if** $i \pmod{g} \equiv 0$ **then**   ▷ Balloon phase
 8:         **for** $j = 1$ to $j = d$ **do**
 9:             $P_j^i = P_{j-1}^i \cup D_j$, where D_j are the nodes at depth d from P_0^i.
10:         $P_{d+1}^i = \mathsf{Valiant}(G_{\leq i}, e, d) \cup \mathsf{PotentialParents}([i+1, i+g])$.   ▷ See Lemma 9
11:     **else**   ▷ Light phase
12:         $P_1^i = P_0^i \cup \{i+1\}$.
13: **else**   ▷ If pre-conditions not met.
14:     **for** $j = 1$ to $j = i+1$ **do**
15:         $P_j^i = P_{j-1}^i \cup \{j\}$.

---

4 k-Restricted Graphs with high CMC

In this section, we describe a construction of k-restricted graphs with high cumulatively memory complexity that builds into our ultimate ciMHF implementation. We first describe the block partition extension gadget, which requires an input graph G and outputs a family of k-restricted dynamic graphs. However, naïvely choosing the input graph G does not yield a construction with high CMC.

Intuitively, the block partition extension gadget takes the last N nodes of G and partitions them into $\frac{N}{k}$ blocks of k nodes each. The gadget then creates N more nodes in a path, such that a parent $r(j)$ of node j in this path is drawn uniformly at random from block i, where $i = j \pmod{\frac{N}{k}}$. The intuition is that by drawing parents uniformly at random from each block in round robin fashion, we encourage an algorithm to keep $\Omega(N)$ nodes on the graph for $\Omega(N)$ steps. Of course, the graph could always maintain $o(n)$ pebbles on the graph and repebble when necessary, but we can discourage this strategy by making the repebbling procedure as expensive as possible.

A first attempt would be to choose a highly depth-robust graph G, such as a grates graph, which informally has long paths of length $\Omega(N^{1-\epsilon})$ for any constant $0 < \epsilon < 1$, even when $\Omega(N)$ nodes are removed from G. Thus if an algorithm does not maintain $\Omega(N)$ pebbles on the graph, the repebbling strategy costs at least $\Omega(N^{2-\epsilon})$. Although this is a good start, this does not quite match the $\Omega(N^2)$ CMC of various dMHFs. We defer full discussion of how to increase the CMC to $\Omega(N^2)$ to later in this section.

Instead, we first define a specific way to obtain a k-restricted dynamic graph given a graph G with N nodes and a parameter k.

▶ **Definition 12** (Block Partition Extension). *Given a DAG $G = (V = [\alpha N], E)$ with αN nodes containing a set of $O = [(\alpha - 1)N + 1, \alpha N]$ output nodes of size N and a parameter k, let $O_i = [(\alpha - 1)N + 1 + ik, (\alpha - 1)N + (i+1)k]$ for $i \in \left[\frac{N}{k}\right]$ so that $\{O_i\}$ forms a partition of O. We define the* block partition extension *of G, denoted $\mathsf{BlockPartition}_k(G)$, as*

a distribution of graphs $\mathbb{G}_{G,k}$. Each graph G' sampled from $\mathbb{G}$ has vertices $V' = [(\alpha+1)N]$ and edges $E' = E \cup F$, where F is defined as the edges $(i-1,i)$ and $(r(i),i)$ for each $i \in [\alpha N+1, (\alpha+1)N]$, where $r(i)$ is drawn uniformly at random from $O_{i \mod \frac{N}{k}}$.

An example of the block partition extension is given in Figure 1.
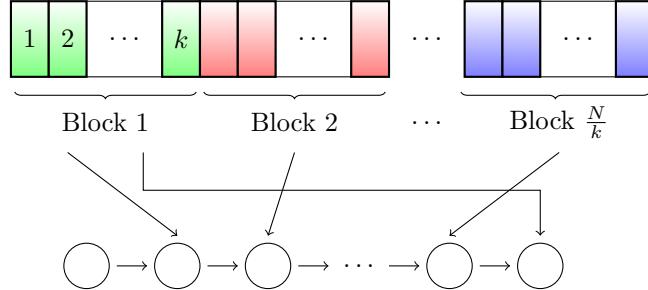
■ **Figure 1** Parent $r(i)$ is drawn uniformly at random from the nodes partitioned to each block.

Our ultimate construction also requires the use of superconcentrator graphs, defined as follows:

▶ **Definition 13.** *A graph G with $\mathcal{O}(N)$ vertices is a* superconcentrator *if there exists an input set I and an output set O with $|I| = |O| = N$ such that for all $S_1 \subseteq I, S_2 \subseteq O$ with $|S_1| = |S_2| = k$, there are k node disjoint paths from S_1 to S_2.*

It is known that there exists superconcentrators with $|I| = |O| = N$, constant indegree and $\mathcal{O}(N)$ total nodes [28, 26]. We now show that a set Y, which contains more nodes than a set S of removed nodes, has at least $N - |S|$ ancestors in $G - S$.

▶ **Lemma 14.** *Given a superconcentrator G with N input nodes I and N output nodes O, let S and $Y \subseteq O$ be sets of nodes with $|S| < |Y|$. Then $|I \cap \mathsf{ancestors}_{G-S}(Y)| \geq N - |S|$.*

Proof. Let $X \subseteq I$ be the last $|Y|$ nodes of I. Since G is a superconcentrator, then G contains at least $|Y|$ node disjoint paths between X and Y. Since $|S| < |Y|$, then one of these paths from X to Y that does not intersect S. Thus, X contains some ancestor of Y in $G - S$ and in fact by considering the paths associated with decreasing order of nodes in X, it follows that $|I \cap \mathsf{ancestors}_{G-S}(Y)| \geq N - |S|$. ◀

We require the use of grates graphs $\{\mathsf{grates}_{N,\epsilon}\}_{N=1}^{\infty}$ [30]. For each constant $\epsilon > 0$ and each $N \geq 1$ the graph $\mathsf{grates}_{N,\epsilon} = (V_N, E_{N,\epsilon})$ has $\mathcal{O}(N)$ nodes and constant indegree $\mathsf{indeg}(\mathsf{grates}_{N,\epsilon}) = \mathcal{O}(1)$. Moreover, the graph $\mathsf{grates}_{N,\epsilon}$ contains source nodes $I_N \subset V_N$ and N sinks $O_N \subset V_N$. Given a set $S \subseteq V_N$ of deleted nodes we say that an output node $y \in O_N$ is c-good with respect to S if $\left| I_N \cap \mathsf{ancestors}_{\mathsf{grates}_{N,\epsilon}-S}(y) \right| \geq cN$ i.e., for at least cN input nodes $x \in I_N$ the graph $\mathsf{grates}_{N,\epsilon} - S$ still contains a path from x to y. The grates graph contains several properties summarized below.

▶ **Theorem 15.** *[30] For each $\epsilon > 0$ there exist constants $\gamma, c > 0$ such that for all $N \geq 1$ the graph $\mathsf{grates}_{N,\epsilon}$ is $(\gamma N, cN^{1-\epsilon})$-depth robust. Furthermore, for each set $S \subseteq V_N$ of size $|S| \leq \gamma N$ at least cN output nodes are still c-good with respect to S. Formally,*

$$\left| \{ x \in O \ : \ |I_N \cap \mathsf{ancestors}_{G-S}(x)| \geq cN \} \right| \geq cN \ .$$

We require the use of graph overlays, defined as follows:
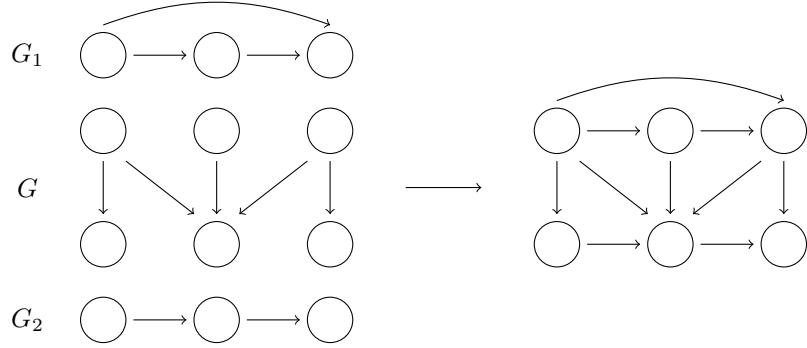
Figure 2 An example of a graph overlay $\mathsf{overlay}(G_1, G, G_2)$.

▶ **Definition 16** (Graph overlays). *Given a DAG $H = (V = [N], E)$ with sources $I = \{1, \ldots, n_1\}$ and sinks $O = \{N - n_2 + 1, \ldots, N\}$, a DAG $G_1 = (V_1 = [n_1], E_1)$, and a DAG $G_2 = (V_2 = [n_2], E_2)$, we define:*

(1) *the graph overlay $G' = \mathsf{overlay}(G_1, H, G_2)$ by $G' = ([N], E')$, where $(i,j) \in E'$ if and only if $(i,j) \in E$ or $(i,j) \in E_1$ or $(i + N - n_2, j + N - n_2) \in E_2$*

(2) *the superconcentrator overlay of an N node DAG G by $\mathsf{superconc}(G) = \mathsf{overlay}(G, \mathsf{SC}_N, L_N)$, where SC_N is a superconcentrator with N input (sources) and output (sinks) nodes and L_N is the line graph of N nodes*

(3) *the grates overlay of an N node DAG G by $\mathsf{grates}_\epsilon(G) = \mathsf{overlay}(G, \mathsf{grates}_{N,\epsilon}, L_N)$.*

An example of a graph overlay is displayed in Figure 2.

We describe a preliminary attempt at a ciMHF construction in Figure 3. At a high level, the construction consists of four components. The first component is a grates graph G_1 with N nodes. The second component is a superconcentrator overlay with $\mathcal{O}(N)$ nodes, including N input nodes and N output nodes, so that $G_2 = \mathsf{superconc}(G_1)$. The third component consists of a grates overlay with $\mathcal{O}(N)$ nodes including N output nodes, so that $G_3 = \mathsf{grates}_\epsilon(G_2)$. The N output nodes of G_3 are partitioned into $\frac{N}{k}$ blocks, each with k nodes, in preparation for a block partition extension in the final component. Namely, the fourth component consists of a k-restricted graph with N nodes, so that $G_4 = \mathsf{BlockPartition}_k(G_3)$.

Sampling Algorithm, for $k = \Omega(N^\epsilon)$:
(1) $G_1 = \mathsf{grates}_{N,\epsilon}$
(2) $G_2 = \mathsf{superconc}(G_1)$
(3) $G_3 = \mathsf{grates}_\epsilon(G_2)$
(4) $G_4 \sim \mathsf{BlockPartition}_k(G_3)$

Figure 3 First attempt at ciMHF. Each parent $r(i)$ is randomly chosen from the labels in specific block corresponding to i.

The intuition for the $\Omega(N^2)$ cumulative pebbling complexity is as follows. Suppose there exists a time t_{bad} with a "small" number of pebbles on the graph. Then with high probability, walking a pebble $s = \frac{N}{4k}$ steps on the final layer of the graph will require some number of output nodes of the grates graph to be repebbled. Again with high probability, repebbling one of these output nodes requires a large number of input nodes of the grates graph to be repebbled. These input nodes are the output nodes of the superconcentrator at the second layer. The superconcentrator property then implies that $\Omega(N)$ nodes of the grates graph

on the first layer will need to be repebbled. For a grates graph that is $(\Omega(N), \Omega(N^{1-\epsilon}))$-depth robust, this cost is at least $\Omega(N^{2-\epsilon})$ every s steps. Thus, the total cost is at least $\min\left(\Omega(N^2), k\Omega(N^{2-\epsilon})\right)$, which is just $\Omega(N^2)$ for $k = \Omega(N^{\epsilon})$.

We now show that our construction in Figure 3 has cumulative memory complexity $\Omega(N^2)$.

▶ **Theorem 17.** *Let G be drawn from the distribution of k-restricted graphs in Figure 3, for $k = \Omega(N^{\epsilon})$. There exist constants $c_1 > 0$ and $c_2 \in (0,1)$ such that for any dynamic pebbling strategy S,*

$$\Pr_{\mathbb{G}}\left[\mathsf{cc}(S,G) > c_1 N^2\right] \geq 1 - c_2^{N/k}.$$

Proof. Let αN be the total number of nodes in G_3 so that the total number of nodes in G is $(\alpha + 1)N$. Let $x, y \in (0,1)$ be constants such that the grates graph G_1 is $(xN, yN^{1-\epsilon})$-depth robust. By Theorem 15, there exist constants $0 < c < \frac{x}{2}$ and $0 < \gamma < c$ such that for any set S with $|S| \leq \gamma N$, at least cN nodes in the output nodes of G_3 are c-good with respect to S. For each node i, let t_i be the first time that node i is pebbled. Suppose there exists a time t_{bad} with $t_i \leq t_{\mathsf{bad}} < t_{i+1}$ such that there are $|P_{t_{\mathsf{bad}}}| < \frac{\gamma N}{4}$ pebbles on the graph.

For a node j in the output set $[(\alpha - 1)N, \alpha N]$ of G_3, we call an index j a *costly index* if j is c-good with respect to $P_{t_{\mathsf{bad}}}$ and let COSTLY be the set of costly indices. Note that if a node $i \in$ COSTLY, then by definition $i \notin P_{t_{\mathsf{bad}}}$. By Theorem 15 and the observation that $|P_{t_{\mathsf{bad}}}| < \frac{\gamma N}{4}$, there are at least cN nodes in the output set of G_3 are c-good with respect to $P_{t_{\mathsf{bad}}}$, i.e., $|\mathsf{COSTLY}| > cN$. Then for $s := \frac{N}{k}$, we call $j \in [i, i+s]$ a *missed costly index* if $r(j) \notin P_{t_{\mathsf{bad}}}$ and let $r(j) \in$ COSTLY.

For each $j \in \left[\frac{N}{k}\right]$, let c_j be the number of costly indices in block j of the output set of G_3, i.e., $c_j := |\mathsf{COSTLY} \cap [(\alpha - 1)N + (j-1)k + 1, (\alpha - 1)N + jk]|$. Since the parents of $[i, i+s]$ are exactly one random node from each of the $\frac{N}{k}$ blocks, then the probability p that no parent of $[i, i+s]$ is a missed costly index is

$$p := \prod_{j=1}^{N/k} \left(1 - \frac{c_j}{k}\right) \leq \left(\frac{k}{N} \sum_{j=1}^{N/k} \left(1 - \frac{c_j}{k}\right)\right)^{N/k},$$

where the inequality holds by the Arithmetic Mean-Geometric Mean Inequality. Since $\sum c_j = cN$, then

$$p \leq \left(1 - \frac{k}{N} \sum_{j=1}^{N/k} \frac{c_j}{k}\right)^{N/k} \leq (1-c)^{N/k}.$$

Thus with high probability, there will be some missed costly index.

By Lemma 14 and the definition of c-good, any missed costly index requires cN nodes in the input set of G_3 to be repebbled. Since the input set of G_3 is connected by a superconcentrator to the output set of G_1, the cN nodes in the input set of G_3 that need to be repebbled have at least $N - cN$ ancestors in the output set of G_1. Thus, at least $N - cN - |P_{t_{\mathsf{bad}}}|$ nodes in G_1 must be repebbled.

Because G_1 is $(xN, yN^{1-\epsilon})$-depth robust, then $G_1 - S$ is $(xN - |S|, yN^{1-\epsilon})$-depth robust for any set S. Moreover, the cost to pebble $G_1 - S$ is at least $(xN - |S|)(yN^{1-\epsilon})$. In particular, if $G_1 - S$ is the set of nodes in G_1 must be repebbled, then it costs at least $(xN - cN - |P_{t_{\mathsf{bad}}}|)(yN^{1-\epsilon})$ to repebble $G_1 - S$. Since $c < \frac{x}{2}$ and $|P_{t_{\mathsf{bad}}}| < \frac{\gamma N}{4} < \frac{cN}{4} < \frac{xN}{8}$, then the cost is at least $\frac{3xy}{8}N^{2-\epsilon}$.

Hence to pebble an interval $[i, i+s]$ with $s = \frac{N}{k}$, either $\frac{\gamma N}{4}$ pebbles are kept on the graph for all $s = \frac{N}{k}$ steps or if we at any point in time $j \in [i, i+s]$ we have $|P_j| \le \gamma N/4$ then (whp) an the pebbling algorithm incurs cost $\frac{3xy}{8} N^{2-\epsilon}$ to repebble the graph during the next s steps $[j, j+s]$. By partitioning the last N nodes of the graph G into k disjoint intervals of length $\frac{N}{k}$, it follows that the total cost is at least $\min\left(\frac{\gamma N^2}{4}, \frac{3}{16}xykN^{2-\epsilon}\right)$. Thus for $k = \Omega(N^\epsilon)$, the total cost is $\Omega(N^2)$ with high probability. ◄

▶ **Corollary 18.** *Let G be drawn from the distribution of k-restricted graphs in Figure 3, for $k = \Omega(N^\epsilon)$. Then $\mathsf{cc}(G) = \Omega(N^2)$.*

5 k-Restricted Graphs: Amenable to Shuffling

In this section, we introduce a useful property for certain dynamic graphs: amenable to shuffling. In Section 6, we will describe computationally data-independent evaluation algorithms for evaluating memory hard function based on dynamic graphs that are amenable to shuffling.

5.1   Characterization of Dynamic Graphs Amenable to Shuffling

We first describe the properties of dynamic graphs that are amenable to shuffling. Recall that for a node i, we define $\mathsf{PotentialParents}(i)$ to be set Y_i of minimal size such that $\mathbf{Pr}\left[r(i) \in Y_i\right] = 1$, where $r(i) < i - 1$ is randomly chosen so that the directed edge $(r(i), i)$ is in the dynamic graph.

▶ **Definition 19** (Amenable to Shuffling). *Let G be a DAG with αN nodes for some constant $\alpha > 1$ and let L be the last N nodes of G. Suppose that L can be partitioned into $\frac{N}{k}$ groups $G_1, \ldots, G_{\frac{N}{k}}$ such that*

(1) *Uniform Size of Groups: $|G_i| = k$ for all $i \in \left[\frac{N}{k}\right]$.*

(2) *Large Number of Potential Parents: For each $v \in L$, $|\mathsf{PotentialParents}(u)| = k$.*

(3) *Potential Parents not in L: For each $v \in L$, $\mathsf{PotentialParents}(u) \subseteq [(\alpha - 1)N]$*

(4) *Same Potential Parents for Each Group: For all $i \in \left[\frac{N}{k}\right]$ and $u, v \in G_i$, $\mathsf{PotentialParents}(u) = \mathsf{PotentialParents}(v)$.*

(5) *Different Potential Parents for Different Groups: For all $i, j \in \left[\frac{N}{k}\right]$ with $i \ne j$, let $u \in G_i$ and $v \in G_j$. Then $\mathsf{PotentialParents}(u) \cap \mathsf{PotentialParents}(v) = \emptyset$.*

(6) *No Collision for Parents: For each $i \in \left[\frac{N}{k}\right]$, define the event UNIQUE_i to be the event that $r(u) \ne r(v)$ for all $u, v \in G_i$. Then $\mathbf{Pr}\left[\mathsf{UNIQUE}_i\right] = 1$ for all $i \in \left[\frac{N}{k}\right]$.*

(7) *Data-Independency: The subgraph induced by the first $(\alpha - 1)N$ nodes is a static graph.*

Then we call G amenable to shuffling.

We shall show in Theorem 24 in Section 6 that dynamic graphs that are amenable to shuffling can be used for memory hard functions with computationally data-independent evaluation algorithms. We now describe a version of Figure 3 that is amenable to shuffling.

5.2   Version of Construction Amenable to Shuffling

To ensure that there does not exist $i \ne j$ such that $r(i) = r(j)$, we slightly modify the construction of block partition extensions to the concept of a collision-resistant block partition extension. For the sake of presentation, note that we use $2N$ output nodes in G in the following definition.

▶ **Definition 20** (Collision-Resistant Block Partition Extension). *Given a DAG $G = (V = [\alpha N], E)$ with αN nodes containing a set of $O = [(\alpha - 2)N + 1, \alpha N]$ output nodes of size $2N$ and a parameter k, let $O_i = [(\alpha - 2)N + 1 + 2ik, (\alpha - 2)N + 2(i + 1)k]$ for $i \in \left[\frac{N}{k}\right]$ so that $\{O_i\}$ forms a partition of O. We define the* collision-resistant block partition extension *of G, denoted $\mathsf{CR} - \mathsf{BlockPartition}_k(G)$, as a distribution of graphs $\mathbb{G}_{G,k}$. Each graph G' sampled from $\mathbb{G}$ has vertices $V' = [(\alpha + 1)N]$ and edges $E' = E \cup F$, where F is defined as the edges $(i - 1, i)$ and $(r(i), i)$ for each $i \in [\alpha N + 1, (\alpha + 1)N]$, where $r(i)$ is defined as follows:*

(1) *Let Enc be the family of all permutations of $[2k]$, so that for each fixed j,*

$$\{\mathsf{Enc}(j, \ell)\}_{\ell \in [2k]} = [2k].$$

(2) *For each $i \in [\alpha N + 1, (\alpha + 1)N]$, let $j = i \mod \frac{N}{k}$ and define $1 \le p \le k$ to be the unique integer such that $i = \frac{N}{k}(p - 1) + \alpha N i + j$. Then we define $r(i) = (\alpha - 2)N + 1 + 2jk + \mathsf{Enc}(x \circ j, p)$, so that $r(i) \in O_j$.*

Observe that the collision-resistant block partition extension is a gadget that yields a dMHF, since the parent function $r(i)$ has the key $x \circ j$ to its permutation function Enc. Hence, the underlying dynamic graph differs across different input values x.

Then our construction of the ciMHF appears in Figure 4 and Figure 5. As before, the construction consists of four layers. The first layer consists of a grates graph with $2N$ nodes, $G_1 = \mathsf{grates}_{2N,\epsilon}$. The second layer consists of a superconcentrator overlay with $\mathcal{O}(N)$ nodes with $2N$ input nodes and $2N$ output nodes, so that $G_2 = \mathsf{superconc}(G_1)$. The third layer consists of a grates overlay with $\mathcal{O}(N)$ nodes including $2N$ output nodes, so that $G_3 = \mathsf{grates}_\epsilon(G_2)$. The $2N$ output nodes of G_3 are partitioned into $\frac{N}{k}$ blocks, each with $2k$ nodes, which allows the final layer to be a $2k$-restricted grpah. In particular, the fourth layer uses a collision-free block partition extension rather than the block partition extension of Figure 3.

Sampling algorithm, for $k = \Omega(N^\epsilon)$:
(1) $G_1 = \mathsf{grates}_{2N,\epsilon}$
(2) $G_2 = \mathsf{superconc}(G_1)$
(3) $G_3 = \mathsf{grates}_\epsilon(G_2)$
(4) $G_4 \sim \mathsf{CR} - \mathsf{BlockPartition}_k(G_3)$

■ **Figure 4** Second attempt at ciMHF. Each parent $r(i)$ is chosen by a permtuation of the labels in specific block corresponding to i. The underlying graph is visualized in Figure 5.

We now show that the construction of Figure 4 has cumulative cost $\Omega(N^2)$ with high probability. The proof is almost verbatim to Theorem 17 except that the graph overlays now have $2N$ input and output nodes.

▶ **Theorem 21.** *Let $0 < \epsilon < 1$ be a constant and $k = \Omega(N^\epsilon)$. Let $\mathbb{G}$ be drawn from the distribution of $2k$-restricted graphs in Figure 4. There exist constants $c_1 > 0$ and $c_2 \in (0, 1)$ such that for any dynamic pebbling strategy S,*

$$\Pr_{G \in \mathbb{G}} \left[\mathsf{cc}(S, G) > c_1 N^2\right] \ge 1 - c_2^{N/k}.$$

▶ **Corollary 22.** *Let $\mathbb{G}$ be drawn from the distribution of $2k$-restricted graphs in Figure 4, for $k = \Omega(N^\epsilon)$. Then $\mathsf{cc}(\mathbb{G}) = \Omega(N^2)$.*

Finally, we observe that the construction of Figure 4 is amenable to shuffling since it satisfies the properties of Definition 19.
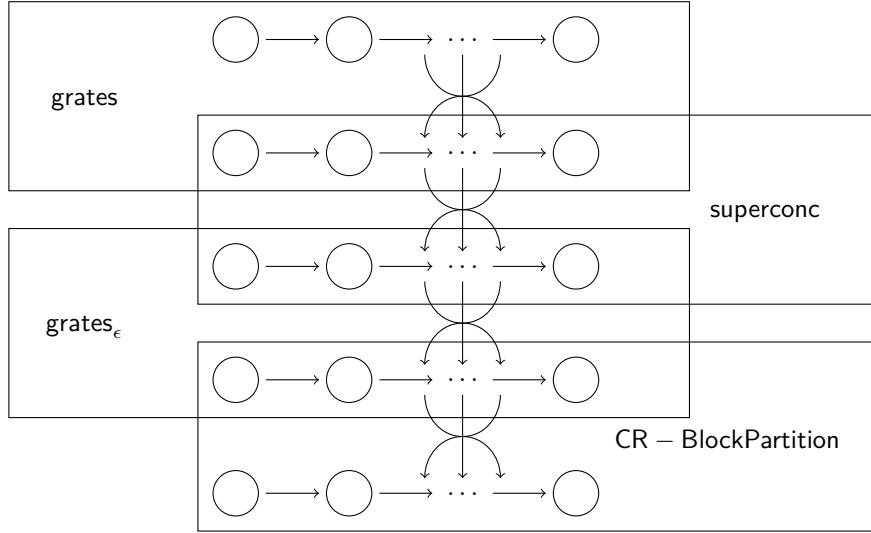
Figure 5 Final construction of Figure 4.

6 Implementation of ciMHF

In this section, we describe how to implement our construction in a way that is computationally data-independent. We first formalize the notion of computationally data-independent and then describe the system model we utilize.

6.1 Computationally Data-Independent MHF (ciMHF) and Systems Model

We define the security of a computationally data-independent memory hard function in terms of the following game: a side-channel attacker $\mathcal{A}$ selects two inputs x_0, x_1 and sends these inputs to an honest party $\mathcal{H}$. We first require the following definition of leakage patterns.

Leakage Pattern

We define the leakage pattern of an evaluation algorithm MHF.Eval by the sequence of request and store instructions made in each round. Specifically, in each round r, an attacker can observe from the leakage pattern the blocks of memory to be loaded into cache, as requested by MHF.Eval. Let $i = (i_1, \ldots, i_m)$ be the sequence of locations of all blocks requested by MHF.Eval in a particular round r through some command $\mathsf{load}(i)$. If i is completely contained in cache, then no events will be observed by the attacker. Otherwise, if i is not completely contained in cache, we use $\mathsf{request}_r$ to denote the locations of the blocks in memory, as well as their sizes, requested by MHF.Eval in round r. Similarly, we use store_r to denote the locations of the blocks, as well as their sizes, stored into memory by MHF.Eval in round r. We do not allow the attacker to observe the contents of the requested or stored blocks. Formally, the leakage pattern LP is the information $\{(\mathsf{request}_r, \mathsf{store}_r)\}_{r=1}^t$ and is dependent on the algorithm MHF.Eval, random oracle H, internal randomness R, and input value x.

Computationally Data-Independency Game

$\mathcal{H}$ runs a (randomized) evaluation algorithm MHF.Eval on both inputs x_0 and x_1, yielding two leakage patterns LP_0 and LP_1, where LP_i for $i \in \{0,1\}$ depends on both the input x_i and the random coins selected during the execution of MHF.Eval. $\mathcal{H}$ then picks a random challenge bit $b \in \{0,1\}$ and sends $\mathsf{LP}_b, \mathsf{LP}_{1-b}$ to $\mathcal{A}$ to simulate a side-channel. The goal of $\mathcal{A}$ is to predict b i.e., match each input with the corresponding leakage pattern. For a secure ciMHF we guarantee that *any* PPT side-channel attacker $\mathcal{A}$ wins the game with only negligible advantage over random guessing.

Formally, the game consists of three phases **setup**, **challenge**, and **guess**, which are described as follows.

Data independency game for ciMHF:

setup In this phase, $\mathcal{A}$ selects the security parameter λ and two challenge messages x_0 and x_1 and sends them to $\mathcal{H}$. Here we assume without loss of generality that the runtime of the evaluation algorithm MHF.Eval on x_0 and x_1 are the same.

challenge In this phase, $\mathcal{H}$ selects a random bit $b \in \{0,1\}$ and random coins $R_0, R_1 \in \{0,1\}^\lambda$ uniformly at random and then samples $\mathsf{lp}_0 \leftarrow \mathsf{LP}(\mathsf{MHF.Eval}(x_0; R_0))$ and $\mathsf{lp}_1 \leftarrow \mathsf{LP}(\mathsf{MHF.Eval}(x_1; R_1))$. $\mathcal{H}$ sends the ordered pair $(\mathsf{lp}_b, \mathsf{lp}_{1-b})$ to $\mathcal{A}$.

guess After receiving $(\mathsf{lp}_b, \mathsf{lp}_{1-b})$, the adversary $\mathcal{A}$ outputs b' as a guess for b. The adversary wins the game if $b = b'$.

The advantage of the adversary to win the game of computationally data independency of the given MHF is defined as

$$Adv_{\mathcal{A},\mathsf{MHF}}^{\mathsf{ind-lp-iMHF}} = \left| \frac{1}{2} - \Pr[\mathcal{A}(x_0, x_1, \mathsf{lp}_b, \mathsf{lp}_{1-b}) = b' : b = b')] \right|,$$

where $\mathsf{lp}_i = \mathsf{LP}(x_i; \mathsf{MHF.Eval}(x_i; R_i))$.

▶ **Definition 23** (Computational data independency). *An evaluation algorithm* MHF.Eval *is computationally data independent if for all non-uniform circuits* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$, *there is a negligible function* $\mathsf{negl}(\cdot)$ *such that* $Adv_{\mathcal{A},\mathsf{MHF}}^{\mathsf{ind-lp-iMHF}} < \mathsf{negl}(\lambda)$.

With the proper random coins, a memory-hard function with an evaluation algorithm that satisfies the above definition reveals only a negligible amount of information through its leakage patterns, and we thus call such a function a computationally data-independent memory hard function.

On the Definition of Computational Data Independency

In section Section 6.2.2 we show that the definition is equivalent to a multi round version of the game in which the attacker can adaptively select the challenge $x_{i,0}, x_{i,1}$ in each round $i \leq r$ after observing $\mathsf{lp}_{i-1,b}, \mathsf{lp}_{i-1,1-b}$ – the memory access patterns from the last round. We also prove that the two security notions are asymptotically equivalent when the attacker runs in polynomial time – in terms of concrete security parameters we lose a factor of r (number of challenge rounds) in the reduction.

We are primarily motivated by the password hashing application where the inputs x_0 and x_1 come from a small domain, as user selected passwords tend to have low entropy [18]. In practice it is reasonable to assume that r is polynomial i.e., if the user only authenticates

poly(λ) times then there are at most $r = $ poly(λ) memory access patterns for the attacker to observe. Assuming that the input domain has size poly(λ) a brute-force attacker cannot use the leaked memory access pattern on input x to eliminate any candidate password x' with high probability, otherwise the attacker could have used the pair x and x' to win the data independency game.

However, in settings where the input domain is very large and r is super-polynomial it will be better to adopt a concrete security definition (see Section 6.2.2). The asymptotic definition in Definition 23 does not definitively rule out the possibility that an attacker can substantially narrow the search space after many (super-polynomial) side channel attacks. For example, suppose that the attacker gets to observe $\mathsf{lp}_i \leftarrow \mathsf{LP}(\mathsf{MHF.Eval}(x; R_i))$ for $i = 1, \ldots, 2^\lambda$, i.e., 2^λ independent evaluations of MHF on secret input x. Supposing that $Adv_{\mathcal{A},\mathsf{MHF}}^{\mathsf{ind}-\mathsf{lp}-\mathsf{iMHF}} = 2^{-\lambda}$ and that the input domain for MHF has size $2^{2\lambda}$, it is possible that each lp_i allows the attacker to eliminate a random subset of $Adv_{\mathcal{A},\mathsf{MHF}}^{\mathsf{ind}-\mathsf{lp}-\mathsf{iMHF}} \times 2^{2\lambda} = 2^\lambda$ candidate inputs, allowing the attacker to find x after just $\mathcal{O}\left(\lambda 2^\lambda\right)$ examples. However, in practice it will usually be reasonable to assume that the attacker gets to observe lp_i a polynomial number of times i.e., the honest party will execute $\mathsf{LP}(\mathsf{MHF.Eval}(x; R_i))$ at most poly(λ) times.

Memory Architecture Assumptions

We consider a tiered random access memory architecture with main memory (RAM) and working memory (cache). We assume that main memory (RAM) is a shared resource with other untrusted processes, each of which have their own cache. Although the operating system kernel will enforce memory separation, i.e., only our program has some region of memory and that other processes cannot read/write to this block, it is also possible that an untrusted process will be able to infer the memory address of read/write operations in RAM (due to side-channel effects).

Formally, the system allows programs access to two operations $\mathsf{Write}(i, x)$, which takes an address i within the memory allocated to the program and writes the value x at address i, and $\mathsf{Read}(i)$ which loads the data at location i. When an operation requests memory at location i, there are two possible outcomes. Either the data item is already in cache or the data item is not in cache. In the second case, the location of the item in memory is revealed through the leakage pattern. Hence, the leakage pattern is either $\perp$, if the data item is already in cache, or i, if the data item is not in cache.

Cache Replacement Policy

We now show that our implementation of the dynamic pebbling construction with cumulative memory cost $\Omega(N^2)$ is computationally data-independent. In particular, we provide an evaluation algorithm whose leakage pattern is computationally indistinguishable under each of the following cache replacement policies:

Least recently used (LRU) This policy tracks the most recent time each item in cache was used and discards the least recently used items first when cache is full and items need to be replaced.

First in first out (FIFO) This policy evicts the first item that was loaded into cache, ignoring how recent or how often it has been accessed.

6.2   ciMHF Implementation

Recall from the definition of a graph labeling in Definition 6 that given a function H and a distribution of dynamic graphs $\mathbb{G}$, the goal is to compute $f_{\mathbb{G},H}(x)$ for some input x, which is equivalent to $f_{G,H}(x)$ once the graph G has been determined by the choice of H and x. In this section, we describe a computationally data-independent implementation of the construction of Section 5.2.

We implement the first three layers as data-independent components. Namely, the grates graph G_1, its superconcentrator overlay G_2 and the subsequent grates overlay G_3 can be implemented deterministically. Observe that G_3 has αN nodes, including $2N$ output nodes O that are partitioned into $\frac{N}{k}$ blocks of size $2k$ each. Specifically $O = O_1 \cup \ldots \cup O_{\frac{N}{k}}$, where $O_j = [(\alpha - 2)N + 1 + 2jk, (\alpha - 2)N + 2(j+1)k]$ for $j \in \left[\frac{N}{k}\right]$.

As stated in Figure 4, the collision-resistant block partition extension G_4 is actually data-independent, since for each $i \in [(\alpha - 1)N, \alpha N]$, each parent $r(i)$ of i is chosen uniformly at random from k possible nodes, but the random procedure is independent of the input x. Hence, the challenge is to implement a computationally data-independent version of the collision-resistant block partition extension. We demand the input of a key K for each computation of $f_{G,H}(x)$. The value of $f_{G,H}(x)$ remains the same across all keys K but the leakage pattern is different for each K.

Data-Dependent Dynamic Graph

For each $i \in [\alpha N + 1, (\alpha + 1)N]$, let $j = i \mod \frac{N}{k}$. To implement a computationally data-independent version of G_4 from Figure 4, we use the value of $L_{G,H,x}(i-1)$ to select a previously "unused" node of O_j as the parent $r(i)$ for i that is not $i - 1$. Here, we say a node $v \in O_j$ is unused if it is not the parent of any node besides $v + 1$ and i.

Since $L_{G,H,x}(i-1)$ can be viewed as a random integer modulo $2k$, we can use $L_{G,H,x}(i-1)$ $(\mod 2k)$ as an input to a permutation with key K to randomly choose the parent of i from O_j. Observe that $i = \alpha N + j$ is the first time a parent will be selected from O_j. Moreover, observe that $m = L_{G,H,x}(i-1) \pmod{2k} + 1$ can be viewed as a random number from $[2k]$ so we set $r(i) = m + (\alpha - 2)N + 1 + 2jk$ as the mth entry of O_j.

Now if $L_{G,H,x}(i-1) \pmod{2k}$ were all unique across the values of $\{i | i \mod \frac{N}{k} \equiv j\}$, then there would be no collisions among selections of parents in O_j and we would be done. However, since these values are not unique, we must do a little more work to avoid collisions, which would reveal information through leakage patterns about the parents of two nodes being the same. To ensure there are no collisions in O_j among parents, we store an array U_j of size $2k$ for each block O_j. For each $1 \le \ell \le 2k$, we initialize $U_j[\ell] = \ell$. The purpose of the U_j array is to ensure that the nodes of O_j that have already appeared as parents are at the end of U_j. In the above example when $r(i) = m + (\alpha - 2)N + 1 + 2jk$, we then set $U_j[m] = 2k$ and $U_j[2k] = m$. Then in the next round of selecting a parent from O_j, we choose uniformly at random from the first $2k - 1$ entries of U_j and in general, for the sth round of selecting a parent from O_j, we choose uniformly at random from the first $2k - s + 1$ entries of U_j.

Specifically for some $2 \le s \le k$, consider the sth iteration in which a parent is selected from O_j. That is, for $i = \alpha N + j + (s - 1)\frac{N}{k}$, the parent $r(i)$ is the sth parent among the nodes of O_j. Observe that $m = L_{G,H,x}(i-1) \pmod{2k - s + 1}$ can be viewed as a random number from $[2k - s + 1]$ and so $U_j[m]$ is a random entry among the unselected $2k - s + 1$ nodes of O_j. We then swap the values of $U_j[m]$ and $U_j[2k - s + 1]$ so that if $U_j[m] = a$ and $U_j[2k - s + 1] = b$ previously then we set $U_j[m] = b$ and $U_j[2k - s + 1] = a$. Hence, the invariant remains that the first $2k - s$ locations of U_j have been unused.
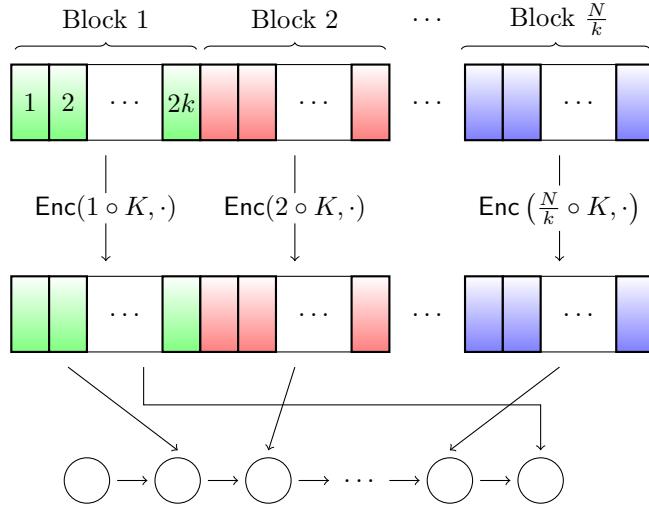
■ **Figure 6** Parent $r(i)$ is drawn uniformly at random from the nodes partitioned to each block.

Shuffling Leakage Patterns

Finally, we point out that the leakage pattern across all computations of $f_{G,H}(x)$ is still the same, since we have not actually incorporated the key K in any of the above details. In summary, the above the description ensures a collision-resistant block partition extension that is data-dependent, but is still vulnerable to side-channel attacks. Hence, we add a final element to our implementation that shuffles the locations of each node $p \in O_j$ inside O_j. That is, for each $1 \leq j \leq \frac{N}{k}$, we use the keyed permutation Enc to store the label of $p \in O_j$ in the location that corresponds to $\mathsf{Enc}(j \circ K, p)$ instead. Thus if $r(i) = p \in O_j$ for some node i, the algorithm must look at the location associated with $\mathsf{Enc}(j \circ K, p)$ to learn the value of $L_{G,H,x}(p)$. Therefore, the underlying graph G is a dynamic graph that is data-dependent but the leakage pattern across each computation of $f_{G,H}(x)$ is different due to the choice of K that shuffles the locations of all labels in each block O_j. A high level example of this shuffling is shown in Figure 6.

Observe that this shuffling must be done completely in the cache to avoid leaking locations of labels during the shuffling. Hence, we require cache eviction policies such as the least recently used (LRU) or first in first out (FIFO) cache eviction policies to ensure that the entire block O_j will remain in cache as the shuffling is performed. We describe the implementation in full in Figure 7.

A Note on Oblivious RAM

The complications with the cache eviction policies and shuffling leakage patterns originate from the necessity of not divulging information in the data-independency game. One reasonable question is whether these complications can be avoided with other implementations that conceal the leakage patterns. For example, an algorithm using oblivious RAM (ORAM), introduced by Goldreich and Ostrovsky [23], reveals no information through the memory access patterns about the underlying operations performed. Thus, an algorithm using an ORAM data structure to evaluate a memory hard function would induce a computationally independent memory hard function, regardless of whether the underlying function is data-

---

Computationally data-independent sequential evaluation algorithm MHF.Eval$(x; R)$ to compute $f_{G,H}(x)$ for any k-restricted dynamic graph G that is amenable to shuffling.

(1) Data-independent phase:

   a. Let G be a k-restricted dynamic graph with αN nodes for some constant $\alpha > 1$ that is amenable to shuffling, L be the last N nodes of G, and H be an arbitrary hash function.

   b. Let $K = \mathsf{Setup}(1^\lambda; R)$ be a hidden random permutation key for each computation of $f_{G,H}(x)$, given the security parameter λ and random bits R.

   c. Recall that the subgraph induced by the first $(\alpha - 1)N$ nodes is a static graph. Compute the label $L_{G,H,x}(v)$ for each node $v \in (G - L)$.

(2) Shuffling phase:

   a. Since G is amenable to shuffling, L can be partitioned into groups $G_1, \ldots, G_{\frac{N}{k}}$ that satisfy the definition of Definition 19. For each $j \in \left[\frac{N}{k}\right]$, let $O_j = \mathsf{PotentialParents}(G_j)$.

   b. For each $j \in \left[\frac{N}{k}\right]$, shuffle the contents of O_j:

     i. Let $v_1, \ldots, v_k$ be the vertices in O_j.

     ii. Load the labels $L_{G,H,x}(v_1), \ldots, L_{G,H,x}(v_k)$ into cache.

     iii. Shuffle the positions of $L_{G,H,x}(v_1), \ldots, L_{G,H,x}(v_k)$ so that for each $p \in [k]$, $L_{G,H,x}(v_p)$ is in the location that previously corresponded to $L_{G,H,x}(v_q)$, where $q = \mathsf{Enc}(j \circ K, p)$, where Enc is a keyed pseudorandom permutation of k.

(3) Data-dependent phase:

   a. For each $j \in \left[\frac{N}{k}\right]$, initialize an array U_j such that for all $1 \le \ell \le k$, $U_j[\ell] = \ell$.

   b. For each $i = \alpha N + 1$ to $(\alpha + 1)N$:

     i. Let j and s be defined so that $1 \le s \le k$ and $1 \le j \le \frac{N}{k}$ given $i = \alpha N + j + (s-1)\frac{N}{k}$ and let $m = L_{G,H,x}(i-1) \pmod{k - s + 1} + 1$.

     ii. Set $r(i) = U_j[m] + (\alpha - 1)N + 1 + jk$ so that $r(i) \in O_j$ and load $L_{G,H,x}(r(i))$. (Recall that the label of $r(i)$ is actually located at the position where the label of node $\mathsf{Enc}(j \circ K, U[m])$ was previously located prior to the shuffling.)

     iii. Load $L_{G,H,x}(r(i))$ and $L_{G,H,x}(i-1)$ and compute $L_{G,H,x}(i) = H(i \circ L_{G,H,x}(r(i)) \circ L_{G,H,x}(i-1))$.

     iv. Let $U_j[U_j[m]] = a$ and $U_j[k - s + 1] = b$. Then swap the values of U_j at $U_j[m]$ and $k - s + 1$ so that $U_j[U_j[m]] = b$ and $U_j[k - s + 1] = a$.

■ **Figure 7** Description of evaluation algorithm for k-restricted graphs that are amenable to shuffling. Note that each computation of $f_{G,H}(x)$ requires as input random bits R to generate the leakage patterns.

dependent or data-independent. [23] describe an oblivious RAM simulator that transforms any program in the standard RAM model into a program in the oblivious RAM model, where the leakage pattern is information theoretically hidden, which is ideal for the data-independency game.

Existing constructions of ORAM protocols such as Path ORAM [31] require amortized $\Omega(\log N)$ bandwidth overhead. Hence given any dMHF and evaluation algorithm running in sequential time M, we can use ORAM to develop a new evaluation algorithm with a concealed leakage pattern, running in sequential time $N = M \log M$. However, this is not ideal because the cumulative memory complexity of the dMHF is $\mathcal{O}\left(M^2\right) = \mathcal{O}\left(\frac{N^2}{\log^2 N}\right)$. Viewed in this way, the ciMHF construction is worse than known iMHF constructions that achieve CMC $\Omega\left(\frac{N^2}{\log N}\right)$ such as DRSample [4, 11]. In fact, even for k-restricted graphs,

we still obtain a blow-up of $\Omega(\log^2 K)$, which is $\Omega\left(\frac{\log^2 N}{\log^2 \log N}\right)$ when $k = \Omega(N^{1/\log\log N})$. Otherwise for $k = o(N^{1/\log\log N})$, our dynamic pebbling attack in Corollary 11 shows that the CMC is at most $o(N^2)$.

Although Boyle and Naor [20] proposed the notion of *online* ORAM, where the operations to be performed arrive in an online manner, and observe that the lower bounds of [23] do not hold for online ORAM, Larsen and Nielsen [25] answer this open question by proving an amortized $\Omega(\log N)$ bandwidth overhead lower bound on the bandwidth of any online ORAM. Therefore, it does not seem obvious how to use ORAM in the implementations of maximally hard ciMHFs.

6.2.1 Implementation and Analysis

Hybrid:

(1) Data-independent phase:
    a. Let G be a k-restricted dynamic graph with αN nodes for some constant $\alpha > 1$ that is amenable to shuffling, L be the last N nodes of G, and H be an arbitrary hash function.
    b. Let $K = \mathsf{Setup}(1^\lambda; R)$ be a hidden random permutation key for each computation of $f_{G,H}(x)$, given the security parameter λ and random bits R.
    c. Recall that the subgraph induced by the first $(\alpha - 1)N$ nodes is a static graph. Compute the label $L_{G,H,x}(v)$ for each node $v \in (G - L)$.

(2) Shuffling phase:
    a. Since G is amenable to shuffling, L can be partitioned into groups $G_1, \ldots, G_{\frac{N}{k}}$ that satisfy the definition of Definition 19. For each $j \in \left[\frac{N}{k}\right]$, let $O_j = \mathsf{PotentialParents}(G_j)$.
    b. For each $j \in \left[\frac{N}{k}\right]$, shuffle the contents of O_j:
        i. Let $v_1, \ldots, v_k$ be the vertices in O_j.
        ii. Load the labels $L_{G,H,x}(v_1), \ldots, L_{G,H,x}(v_k)$ into cache.
        iii. Shuffle the positions of $L_{G,H,x}(v_1), \ldots, L_{G,H,x}(v_k)$ so that for each $p \in [k]$, $L_{G,H,x}(v_p)$ is in the location that previously corresponded to $L_{G,H,x}(v_q)$, where $q = \mathsf{Enc}(j \circ K, p)$, where Enc is a keyed truly random permutation of k.

(3) Data-dependent phase:
    a. For each $j \in \left[\frac{N}{k}\right]$, initialize an array U_j such that for all $1 \leq \ell \leq k$, $U_j[\ell] = \ell$.
    b. For each $i = \alpha N + 1$ to $(\alpha + 1)N$:
        i. Let j and s be defined so that $1 \leq s \leq k$ and $1 \leq j \leq \frac{N}{k}$ given $i = \alpha N + j + (s-1)\frac{N}{k}$ and let $m = L_{G,H,x}(i-1) \pmod{k-s+1} + 1$.
        ii. Set $r(i) = U_j[m] + (\alpha - 1)N + 1 + jk$ so that $r(i) \in O_j$ and load $L_{G,H,x}(r(i))$. (Recall that the label of $r(i)$ is actually located at the position where the label of node $\mathsf{Enc}(j \circ K, U[m])$ was previously located prior to the shuffling.)
        iii. Load $L_{G,H,x}(r(i))$ and $L_{G,H,x}(i-1)$ and compute $L_{G,H,x}(i) = H(i \circ L_{G,H,x}(r(i)) \circ L_{G,H,x}(i-1))$.
        iv. Let $U_j[U_j[m]] = a$ and $U_j[k-s+1] = b$. Then swap the values of U_j at $U_j[m]$ and $k-s+1$ so that $U_j[U_j[m]] = b$ and $U_j[k-s+1] = a$.

🟨 **Figure 8** Description of hybrid. Differs from Figure 7 in that the hidden input key is used to index into the entire family of random permutations, rather than a pseudorandom permutation.

We require the hybrid in Figure 8 to argue that our implementation of Figure 4 is a ciMHF. The hybrid in Figure 8 differs from the implementation of Figure 4 in Figure 7 in that the hidden input key is used to index from the entire family of random permutations, rather than a pseudorandom permutation. Thus the only way an adversary can distinguish between the hybrid and the real world sampler is by distinguishing between a random permutation and a pseudorandom permutation. On the other hand, if an adversary fails to distinguish between the hybrid and the real world sampler, then the cumulative memory complexity of the implementation requires $\Omega(N^2)$ since the leakage pattern of the hybrid is statistically equivalent to the dMHF construction in Figure 4, where each parent is chosen a priori using a permutation drawn uniformly at random.

▶ **Theorem 24.** *For each DAG G that is amenable to shuffling, there exists a computationally data-independent sequential evaluation algorithm* $\mathsf{MHF.Eval}(x; R)$ *computing the function* $f_{\mathbb{G},H}$ *in time* $\mathcal{O}(N)$.

Proof. Consider the evaluation function in Figure 7. Observe that the hybrid in Figure 8 has the same distribution of leakage patterns as the dMHF of Figure 4. Moreover, under the least recently used (LRU) or first in first out (FIFO) cache eviction policies, if k is less than the size of the cache, then all the shuffling can be performed so an attacker observing the leakage patterns of the hybrid has no advantage in the data-independency game. Furthermore, the ciMHF implementation in Figure 7 only differs from the hybrid in Figure 8 in the implementation of Enc as a pseudorandom permutation compared to a truly random permutation. Therefore, an attacker observing leakage patterns from the implementation in Figure 7 only obtains a negligible advantage $\mathsf{negl}(\lambda)$ in the security parameter λ, in the data-independency game. Hence, the implementation of Figure 7 is a ciMHF. ◀

We now show that the evaluation function in Figure 7 of the dMHF in Figure 4 is a maximally hard ciMHF.

▶ **Theorem 25.** *Let $0 < \epsilon < 1$ be a constant and $k = \Omega(N^\epsilon)$. Then there exists a family $\mathbb{G}$ of k-restricted graphs with $\mathsf{cc}(\mathbb{G}) = \Omega(N^2)$ that is amenable to shuffling. Moreover, there exists a negligible value $\delta = \mathsf{negl}(N)$ such that $\mathsf{cc}_\delta(\mathbb{G}) = \Omega(N^2)$.*

Proof. Consider the evaluation function in Figure 7 of the dMHF in Figure 4. For the sake of completeness, the full implementation is also shown in Figure 9. Since the construction of Figure 4 is amenable to shuffling, then the evaluation algorithm is a ciMHF by Theorem 24. Finally by Theorem 21, $\mathsf{cc}(\mathbb{G}) = \Omega(N^2)$.

In fact, Theorem 21 implies that for $G \in \mathbb{G}$ drawn uniformly at random and any pebbling strategy S, not only is $\mathbb{E}_{G \sim \mathbb{G}}[\mathsf{cc}(S, G)] = \Omega(N^2)$, but also $\mathsf{cc}(S, G) = \Omega(N^2)$ with probability at least $1 - c^{N/k}$ for some constant $0 < c < 1$. Thus for $\delta = 1 - c^{N/k}$, we have $\mathsf{cc}(S, \mathbb{G}, \delta) = \Omega(N^2)$ for any pebbling strategy S and so $\mathsf{cc}_\delta(\mathbb{G}) = \Omega(N^2)$. ◀

For the sake of completeness, we give the evaluation algorithm for the maximally hard ciMHF in Figure 9.

Computationally data-independent sequential evaluation algorithm $\mathsf{MHF.Eval}(x; R)$

(1) Let H be an arbitrary hash function and $K = \mathsf{Setup}(1^\lambda; R)$ be a hidden random permutation key for each computation of $f_{\mathbb{G}, H}(x)$, given the security parameter λ and random bits R. Let $k = \Omega(N^\epsilon)$.

(2) Data-independent phase:
   a. $G_1 = \mathsf{grates}_{2N, \epsilon}$
   b. $G_2 = \mathsf{superconc}(G_1)$
   c. $G_3 = \mathsf{grates}_\epsilon(G_2)$, which has αN total nodes, including $2N$ output nodes $O = O_1 \cup \ldots \cup O_{\frac{N}{k}}$, where $O_j = [(\alpha-2)N + 1 + 2ik, (\alpha-2)N + 2(j+1)k]$ for $j \in \left[\frac{N}{k}\right]$.
   d. Compute the label $L_{G_3, H, x}(v)$ for each node $v \in G_3$.

(3) Shuffling phase:
   a. Let G be the graph G_3 appended with N additional nodes, so that $V = [(\alpha+1)N]$, and edges $(i-1, i)$ for each $\alpha N + 1 \leq i \leq (\alpha+1)N$.
   b. For each $j \in \left[\frac{N}{k}\right]$, shuffle the contents of O_j:
      i. Let $v_1, \ldots, v_{2k}$ be the vertices in O_j.
      ii. Load the labels $L_{G, H, x}(v_1), \ldots, L_{G, H, x}(v_{2k})$ into cache.
      iii. Shuffle the positions of $L_{G, H, x}(v_1), \ldots, L_{G, H, x}(v_{2k})$ so that for each $p \in [2k]$, $L_{G, H, x}(v_p)$ is in the location that previously corresponded to $L_{G, H, x}(v_q)$, where $q = \mathsf{Enc}(j \circ K, p)$, where Enc is a keyed pseudorandom permutation of $2k$.

(4) Data-dependent phase:
   a. For each $j \in \left[\frac{N}{k}\right]$, initialize an array U_j such that for all $1 \leq \ell \leq 2k$, $U_j[\ell] = \ell$.
   b. For each $i = \alpha N + 1$ to $(\alpha+1)N$:
      i. Let j and s be defined so that $1 \leq s \leq k$ and $1 \leq j \leq \frac{N}{k}$ given $i = \alpha N + j + (s-1)\frac{N}{k}$ and let $m = L_{G, H, x}(i-1) \pmod{2k - s + 1} + 1$.
      ii. Set $r(i) = U_j[m] + (\alpha-2)N + 1 + 2jk$ so that $r(i) \in O_j$ and load $L_{G, H, x}(r(i))$. (Recall that the label of $r(i)$ is actually located at the position where the label of node $\mathsf{Enc}(j \circ K, U[m])$ was previously located prior to the shuffling.)
      iii. Load $L_{G, H, x}(r(i))$ and $L_{G, H, x}(i-1)$ and compute $L_{G, H, x}(i) = H(i \circ L_{G, H, x}(r(i)) \circ L_{G, H, x}(i-1))$.
      iv. Let $U_j[U_j[m]] = a$ and $U_j[2k - s + 1] = b$. Then swap the values of U_j at $U_j[m]$ and $2k - s + 1$ so that $U_j[U_j[m]] = b$ and $U_j[2k - s + 1] = a$.

■ **Figure 9** Description of implementation of *maximally hard* ciMHF. Again note that each computation of $f_{G, H}(x)$ requires as input random bits R to generate the leakage pattern.

6.2.2   Extension to Multiple Rounds

Finally, we show that our ciMHF implementation is robust to multiple rounds of leakage by considering a data independency game where an adversary is allowed to submit and observe multiple adaptive queries before outputting a guess for the hidden challenge bit b. The game again consists of the phases **setup**, **challenge**, and **guess**, which are described as follows.

> **Adaptive data independency game for ciMHF:**
> **setup** In this phase, $\mathcal{A}$ selects the security parameter λ and sends it to $\mathcal{H}$. $\mathcal{H}$ then selects a random bit $b \in \{0, 1\}$.
> **challenge** For each round $i = 1, 2, \ldots$, $\mathcal{A}$ chooses two adaptive query messages $x_{i,0}$ and $x_{i,1}$ and sends the query messages to $\mathcal{H}$. $\mathcal{H}$ selects random coins $R_{i,0}, R_{i,1} \in \{0,1\}^\lambda$ uniformly at random, samples $\mathsf{lp}_{i,0} \leftarrow \mathsf{LP}(\mathsf{MHF.Eval}(x_{i,0}; R_{i,0}))$ and $\mathsf{lp}_{i,1} \leftarrow \mathsf{LP}(\mathsf{MHF.Eval}(x_{i,1}; R_{i,1}))$, and sends the ordered pair $(\mathsf{lp}_{i,b}, \mathsf{lp}_{i,1-b})$ to $\mathcal{A}$. Note: $\mathcal{A}$ can pick $x_{i+1,0}$ and $x_{i+1,1}$ adaptively after observing the response $(\mathsf{lp}_{i,b}, \mathsf{lp}_{i,1-b})$.
> **guess** The game ends when the adversary $\mathcal{A}$ outputs b' as a guess for b. $\mathcal{A}$ wins the game if $b = b'$.

As before, the advantage of the adversary to win the adaptive data independency game for ciMHF is:

$$Adv_{\mathcal{A},\mathsf{MHF}}^{\mathsf{ind-mult-lp-iMHF}} = \left| \frac{1}{2} - \mathbf{Pr}\left[\mathcal{A}(\mathcal{T}) = b' : b = b' \right] \right|,$$

where $\mathcal{T}$ is the transcript $\{x_{i,0}, x_{i,1}, \mathsf{lp}_{i,b}, \mathsf{lp}_{i,1-b}\}$ and $\mathsf{lp}_{i,j} = \mathsf{LP}(x_{i,j}; \mathsf{MHF.Eval}(x_{i,j}; R_{i,j}))$.

▶ **Definition 26.** *We say an evaluation algorithm* $\mathsf{MHF.Eval}$ *has* (t, ϵ)-*single security if any attacker running in time t has at most advantage ϵ in the data independency game. Similarly, we say an evaluation algorithm* $\mathsf{MHF.Eval}$ *has* (t, r, ϵ)-*adaptive security if any attacker running in time t and making r queries has at most advantage ϵ in the adaptive data independency game.*

We conclude by noting the following relationship between single security and adaptive security, thus implying the security of our evaluation function in Figure 7 of the dMHF in Figure 4 with respect to the adaptive data independency game.

▶ **Theorem 27.** (t, ϵ)-*single security implies* $(t - \mathcal{O}(r \cdot \mathsf{time}(\mathsf{MHF.Eval})), r, r\epsilon)$-*adaptive security.*

Proof. Suppose that $\mathsf{A}_{adaptive}$ violates $(t - \mathcal{O}(r \cdot \mathsf{time}(\mathsf{MHF.Eval})), r, r\epsilon)$-adaptive security for the sake of contradiction. Without loss of generality we will assume that $\mathsf{A}_{adaptive}$ outputs $b' = b$ with probability greater than $\frac{1}{2} + r\epsilon$. We will use $\mathsf{A}_{adaptive}$ to construct an attacker A_{single} that violates (t, ϵ)-single security.

We first define a sequence of r hybrids in the adaptive data-independency game. In Hybrid i, the challenger $\mathcal{H}$ picks bits $b, b_1, \ldots, b_{i-1}$ uniformly at random and sets $b_i = b, b_{i+1} = b, \ldots, b_r = b$. In round j when the attacker $\mathsf{A}_{adaptive}$ submits two strings $x_{j,0}$ and $x_{j,1}$, the challenger $\mathcal{H}$ samples $\mathsf{lp}_{j,0} \leftarrow \mathsf{LP}(\mathsf{MHF.Eval}(x_{j,0}; R_{j,0}))$ and $\mathsf{lp}_{j,1} \leftarrow \mathsf{LP}(\mathsf{MHF.Eval}(x_{j,1}; R_{j,1}))$ and then responds with $\mathsf{lp}_{j,b_i}, \mathsf{lp}_{j,1-b_i}$ instead of $\mathsf{lp}_{j,b}$ and $\mathsf{lp}_{j,1-b}$ i.e., the bit b_i is used to permute the order of the responses in round i instead of b.

Observe that in Hybrid 1, $b_1 = \ldots = b_r = b$, so that Hybrid 1 is equivalent to the actual adaptive independency game. Similarly, in Hybrid r, the bits $b_1, \ldots, b_r$ are all picked independently so that $\mathsf{A}_{adaptive}$ working in Hybrid r has no advantage i.e., the attacker guesses $b' = b$ correctly with probability at most $\mathbf{Pr}\left[b' = b \mid \text{Hybrid } r \right] = \frac{1}{2}$. We observe that the advantage of the attacker is

$$\mathbf{Pr}\left[b' = b \mid \text{Hybrid } 1 \right] - \frac{1}{2} = \mathbf{Pr}\left[b' = b \mid \text{Hybrid } 1 \right] - \mathbf{Pr}\left[b' = b \mid \text{Hybrid } r \right] = \Delta_2 + \ldots + \Delta_r,$$

where $\Delta_i = \mathbf{Pr}\left[b' = b \mid \text{Hybrid } i-1 \right] - \mathbf{Pr}\left[b' = b \mid \text{Hybrid } i \right]$. By an averaging argument, we must have $\Delta_{i+1} > \epsilon$ for some $i < r$. The following observation will also be useful:

$$\Delta_{i+1} = \frac{1}{2}\mathbf{Pr}\left[b' = b \mid \text{Hybrid } i \right] - \frac{1}{2}\mathbf{Pr}\left[b = b' \mid \text{Hybrid } i+1, b_i \neq b \right].$$

Reduction

We now define A_{single} as follows: (1) A_{single} simulates $A_{adaptive}$ along with the adaptive challenger $\mathcal{H}_{adaptive}$. A_{single} generates random bits $b_1, \ldots, b_{i-1}$ and sets $b_{i+1} = \ldots = b_r = b''$ for another random bit b''. In each round $j \neq i$, when $A_{adaptive}$ outputs a query $x_{j,0}, x_{j,1}$, our attacker A_{single} simply computes $lp_{i,0} \leftarrow LP(MHF.Eval(x_{i,0}; R_{i,0}))$ and $lp_{i,1} \leftarrow LP(MHF.Eval(x_{i,1}; R_{i,1}))$ and responds with $lp_{i,b_i}, lp_{i,1-b_i}$. When $A_{adaptive}$ outputs the query $x_{i,0}, x_{i,1}$ in round i, A_{single} forwards this query to the challenger for the single stage challenger $\mathcal{H}_{single}$ and receives back $lp_{i,b}, lp_{i,1-b}$ for an unknown bit b selected by $\mathcal{H}_{single}$. Finally, when $A_{adaptive}$ outputs a guess b' (for b'') A_{single} outputs the same guess b' (for b).

Analysis

Notice that since b'' is just a bit selected uniformly at random and independent from b, then $\mathbf{Pr}\left[b' = b'' \mid b'' = b\right] = \mathbf{Pr}\left[b' = b'' \mid \text{Hybrid } i\right]$. Then from the above observation, we have $\mathbf{Pr}\left[b' = b'' \mid b'' \neq b\right] = \mathbf{Pr}\left[b' = b'' \mid \text{Hybrid } i+1, b_i \neq b''\right] = \mathbf{Pr}\left[b' = b'' \mid \text{Hybrid } i\right] - 2\Delta_{i+1}$. It follows that $\mathbf{Pr}\left[b' = b'' \mid b'' = b\right] - \mathbf{Pr}\left[b' = b'' \mid b'' \neq b\right] = 2\Delta_{i+1}$. Thus, the probability that A_{single} wins is

$$\mathbf{Pr}\left[b' = b\right] = \mathbf{Pr}\left[b'' = b\right]\mathbf{Pr}\left[b' = b'' \mid b'' = b\right] + \mathbf{Pr}\left[b'' \neq b\right]\left(1 - \mathbf{Pr}\left[b' = b'' \mid b'' \neq b\right]\right)$$
$$= \frac{1}{2} + \Delta_{i+1} > \epsilon.$$

Furthermore, the running time of A_{single} is at most t. This contradicts the assumption that the evaluation algorithm MHF.Eval has (t, ϵ)-single security. Therefore, (t, ϵ)-single security implies $(t - \mathcal{O}\left(r \cdot \text{time}(MHF.Eval)\right), r, r\epsilon)$-adaptive security. ◄

───── **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.

2   Joël Alwen and Jeremiah Blocki. Efficiently Computing Data-Independent Memory-Hard Functions. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Proceedings, Part II*, pages 241–271, 2016.

3   Joël Alwen and Jeremiah Blocki. Towards Practical Attacks on Argon2i and Balloon Hashing. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P*, pages 142–157, 2017.

4   Joël Alwen, Jeremiah Blocki, and Benjamin Harsha. Practical Graphs for Optimal Side-Channel Resistant Memory-Hard Functions. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 1001–1017, 2017.

5   Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-Robust Graphs and Their Cumulative Memory Complexity. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part III*, pages 3–32, 2017.

6   Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt Is Maximally Memory-Hard. In *Advances in Cryptology - EUROCRYPT - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part III*, pages 33–62, 2017.

7   Joël Alwen and Vladimir Serbinenko. High Parallel Complexity Graphs and Memory-Hard Functions. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, 2015.

8   Daniel J Bernstein. Cache-timing attacks on AES, 2005.

9 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.

10 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*, pages 292–302, 2016.

11 Jeremiah Blocki, Benjamin Harsha, Siteng Kang, Seunghoon Lee, Lu Xing, and Samson Zhou. Data-Independent Memory Hard Functions: New Attacks and Stronger Constructions. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Proceedings, Part II*, pages 573–607, 2019.

12 Jeremiah Blocki, Benjamin Harsha, and Samson Zhou. On the Economics of Offline Password Cracking. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings*, pages 853–871, 2018.

13 Jeremiah Blocki, Seunghoon Lee, and Samson Zhou. Approximating Cumulative Pebbling Cost is Unique Games Hard. *CoRR*, abs/1904.08078, 2019. `arXiv:1904.08078`.

14 Jeremiah Blocki, Ling Ren, and Samson Zhou. Bandwidth-Hard Functions: Reductions and Lower Bounds. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 1820–1836, 2018.

15 Jeremiah Blocki and Samson Zhou. On the Depth-Robustness and Cumulative Pebbling Cost of Argon2i. In *Theory of Cryptography - 15th International Conference, TCC, Proceedings, Part I*, pages 445–465, 2017.

16 Jeremiah Blocki and Samson Zhou. On the Computational Complexity of Minimal Cumulative Cost Graph Pebbling. In *Financial Cryptography and Data Security - 22nd International Conference, FC, Revised Selected Papers*, pages 329–346, 2018.

17 Dan Boneh, Henry Corrigan-Gibbs, and Stuart E. Schechter. Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part I*, pages 220–248, 2016.

18 Joseph Bonneau. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, San Francisco, CA, USA, May 21–23 2012. IEEE Computer Society Press. `doi:10.1109/SP.2012.49`.

19 Xavier Boyen. Halting Password Puzzles: Hard-to-break Encryption from Human-memorable Keys. In *Proceedings of the 16th USENIX Security Symposium*, 2007.

20 Elette Boyle and Moni Naor. Is There an Oblivious RAM Lower Bound? In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 357–368, 2016.

21 Cynthia Dwork and Moni Naor. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Proceedings*, pages 139–147, 1992.

22 Christian Forler, Stefan Lucks, and Jakob Wenzel. Memory-Demanding Password Scrambling. In *Advances in Cryptology - ASIACRYPT - 20th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part II*, pages 289–305, 2014.

23 Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

24 Marcos A. Simplício Jr., Leonardo C. Almeida, Ewerton R. Andrade, Paulo C. F. dos Santos, and Paulo S. L. M. Barreto. Lyra2: Password Hashing Scheme with improved security against time-memory trade-offs. *IACR Cryptology ePrint Archive*, page 136, 2015.

25 Kasper Green Larsen and Jesper Buus Nielsen. Yes, There is an Oblivious RAM Lower Bound! In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Proceedings, Part II*, pages 523–542, 2018.

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

27 Colin Percival. Stronger key derivation via sequential memory-hard functions, 2009.

28   Nicholas Pippenger. Superconcentrators. *SIAM J. Comput.*, 6(2):298–304, 1977.

29   Ling Ren and Srinivas Devadas. Bandwidth Hard Functions for ASIC Resistance. In *Theory of Cryptography - 15th International Conference, TCC 2017, Proceedings, Part I*, pages 466–492, 2017.

30   Georg Schnitger. On Depth-Reduction and Grates. In *24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 323–328, 1983.

31   Emil Stefanov, Marten van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An Extremely Simple Oblivious RAM Protocol. *J. ACM*, 65(4):18:1–18:26, 2018.

32   Leslie G. Valiant. Graph-Theoretic Arguments in Low-Level Complexity. In *Mathematical Foundations of Computer Science 1977, 6th Symposium, Proceedings*, pages 162–176, 1977.