Memory-Hard Functions from Cryptographic Primitives

Binyi Chen[1(✉)] and Stefano Tessaro[2(✉)]

[1] UC Santa Barbara, Santa Barbara, USA
binyichen@cs.ucsb.edu

[2] University of Washington, Seattle, USA
tessaro@cs.washington.edu

Abstract. Memory-hard functions (MHFs) are moderately-hard functions which enforce evaluation costs both in terms of time and memory (often, in form of a trade-off). They are used e.g. for password protection, password-based key-derivation, and within cryptocurrencies, and have received a considerable amount of theoretical scrutiny over the last few years. However, analyses see MHFs as modes of operation of some underlying hash function $\mathcal{H}$, modeled as a monolithic random oracle. This is however a very strong assumption, as such hash functions are built from much simpler primitives, following somewhat ad-hoc design paradigms.

This paper initiates the study of how to securely instantiate $\mathcal{H}$ within MHF designs using common cryptographic primitives like block ciphers, compression functions, and permutations. Security here will be in a model in which the adversary has parallel access to an idealized version of the underlying primitive. We will provide provably memory-hard constructions from all the aforementioned primitives. Our results are generic, in that we will rely on hard-to-pebble graphs designed in prior works to obtain our constructions.

One particular challenge we encounter is that $\mathcal{H}$ is usually required to have large outputs (to increase memory hardness without changing the description size of MHFs), whereas the underlying primitives generally have small output sizes.

Keywords: Memory-hard functions · Provable security · Ideal models

1 Introduction

Memory-hard functions (MHFs) are functions which are moderately hard to compute both in terms of *time* and *memory*, in the sense that their computation is subject to a time-memory trade-off – relatively fast computation requires memory, whereas low-memory implies slow (or even very slow) computation. This ensures for example that the area-time complexity of custom-made hardware (e.g., ASICs) needed to evaluate MHFs is large (and thus, the dollar cost of this hardware), and this fact makes them suitable for password hashing, password-based key derivation, and proof of work in cryptocurrencies, where attackers may leverage such hardware. The first practical MHF, Scrypt, was proposed by Percival [19,20]. Starting with Alwen and Serbinenko [7], several works have been devoted to the theoretical analysis of MHFs (cf. e.g. [1–8,12–14]), also exposing weaknesses in practical designs like Argon2 [10] (the winner of the password-hashing competition), Catena [17], and Balloon hashing [13].

The starting point of our work is the observation that theoretical works describe MHFs as modes of operation of an underlying primitive, usually a (hash) function $\mathcal{H} : \{0,1\}^M \to \{0,1\}^W$ (where $M \geq W$), modeled as a *random oracle* [9] within security proofs. However, this completely ignores the implementation details behind $\mathcal{H}$ which may make it far from an ideal random oracle – often, such designs are completely ad-hoc and based on much simpler objects (e.g., Scrypt's resembles a permutation-based stream-cipher design), in particular because W is much larger than for conventional hash functions (e.g., a few thousand bits).

Therefore, we would like to study MHFs at a finer level of granularity that considers the inner structure of $\mathcal{H}$, and we would like to understand *how* such an $\mathcal{H}$ is meant to be built in a sound way. We stress that it is not enough for $\mathcal{H}$ to be a random oracle in the sense of *indifferentiability* [18], since memory-hardness definitions are multi-stage games to which indifferentiability does not apply [22]. Therefore, such analyses would call for completely new theory.

We also note that the primitive on which an MHF is based matters – Ren and Devadas [21] pointed out the advantages of building MHFs from AES, as the availability of on-chip hardware implementations (AES-NI) significantly reduces the efficiency speed-up of dedicated hardware by ensuring a conventional CPU can evaluate the function *already* at a cost similar to that of dedicated hardware.

<u>Our contributions – a high-level view.</u> This paper initiates the study of provably-secure MHFs built from basic symmetric primitives, which we model as ideal – we consider *block ciphers*, *permutations* and *compression functions*. We prove general results that will enable us to obtain MHFs based on them, in a model where these primitives are ideal and the adversary is allowed to make multiple calls *in parallel*. (This naturally adapts the parallel random-oracle model from previous MHF analyses to primitives.)

As our first contribution, we provide one-call efficient instantiations of $\mathcal{H}$ from such primitives. We will adapt previous lemmas based on "ex-post-facto arguments" (dating back to [15]) to reduce the security of a large class of MHFs based on directed acyclic graphs (DAGs) to the pebbling complexity of the underlying DAG. (These are usually called data-independent MHFs, or iMHFs for short, and are favored designs due to their resilience to side-channel attacks.)

This will already give us iMHFs from all aforementioned primitives. However, a DAG $\mathbb{G}$ of N vertice yields a function whose computation operates on N memory blocks of size L bits, where L is the output length of the primitive (e.g., $L = 128$ bits for AES). In this case, a good choice of $\mathbb{G}$ would ensure that product of time and memory[1] to evaluate the function is (nearly) $\Omega(N^2 L)$ – increasing memory hardness means increasing N, which leads to a larger function description. A better option (consistent with practice) is to keep the same graph, but operate on *larger* blocks of size $W \gg L$, to ensure the time-memory product is now $\Omega(N^2 W)$.

To do this, we will provide a generic construction of an $\mathcal{H}$ with W-bit output using an underlying primitive with a shorter output. We will refer to $\mathcal{H}$ as a *wide-block labeling function* (as opposed to a *small-block* one like those we gave above), and the resulting MHF will be called a *wide-block MHF*. (Our design will have the added benefit of allowing for a variable W.) Our construction will guarantee the final MHF is memory hard as long as the graph $\mathbb{G}$ is sufficiently *depth-robust*, a notion we review below.

We stress that all practical constructions implicitly design wide-block labeling functions, for which existing analyses provide no guarantees, as they abstract them away as random oracles, which they are not. While we failed to provide either proofs or attacks on practical designs, initiating the study of provably secure constructions in the more realistic primitive-based setting is an important step.

The remainder of this section will provide a more in-detail overview of our results, as well as a concise introduction to the formalism.

1.1 Overview of Our Results

Before highlighting our result in more detail, we briefly review some notions at an informal level.

GRAPH-BASED iMHFs. This paper deals with *graph-based data-independent* MHFs (which we refer to as iMHFs, for short), defined by a DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ on N vertices. For simplicity, we assume $\mathbb{V} = \{v_1, \ldots, v_N\}$, and each edge has the form (v_i, v_j) where $i < j$, i.e., vertices are topologically order – vertex v_1 is the *source*, v_N is the (unique) *sink*. Previous works [3,7] use $\mathbb{G}$ and a *labeling* function $\mathcal{H} : \{0,1\}^{\leq \delta W + \log N} \rightarrow \{0,1\}^W$ (we refer to W as the *block length* of the MHF and δ the maximal indegree of $\mathbb{G}$) to instantiate an MHF $\mathcal{F}_{\mathbb{G},\mathcal{H}}$. On input M, we first assign the label $\ell_1 = \mathcal{H}(\langle 1 \rangle \| M)$ to the source, where $\langle i \rangle$ is a $O(\log N)$-bit encoding of i, and then each vertex v_i is assigned a label

$$\ell_i = \mathcal{H}(\langle i \rangle \| \ell_{j_1} \| \ldots \| \ell_{j_d}) ,$$

where $v_{j_1}, \ldots, v_{j_d}$ are the predecessor vertices of v_i. Finally, we output ℓ_N.

CMC AND PEBBLING. To capture the evaluation costs for $\mathcal{F}_{\mathbb{G},\mathcal{H}}$, following [7] we adopt the *cumulative memory complexity* (CMC). We model the labeling function $\mathcal{H}$ as a random oracle, and assume the adversary proceeds in steps. In

[1] We will use the more fine-grained metric of cumulative memory complexity (CMC), but for now the product of time and memory will suffice for an informal understanding.

each step i, the adversary holds state σ_{i-1} (where $\sigma_0 = M$), and can compute a *vector* of queries to $\mathcal{H}$, as well as next state σ_i, which it will receive in the next step, together with the outputs of the evaluations of $\mathcal{H}$ on the query vector. The CMC of the adversary is defined as the sum of the sizes of states, and $\mathsf{CMC}(\mathcal{F}_{G,\mathcal{H}})$ denotes the best-possible (expected) CMC of an adversary to evaluate $\mathcal{F}_{G,\mathcal{H}}$.

The evaluation of $\mathcal{F}_{\mathbb{G},\mathcal{H}}$ is tightly related to a *(parallel) pebbling game* for the graph $\mathbb{G}$. Initially, the graph has no pebble on it, but in each step i, the player can (i) remove any subset of the pebbles, and (ii) put a pebble on a vertex $v \in \mathbb{V}$ if all parents of v have been pebbled at the previous step. (This is vacuously true for the source.) Multiple legal moves can be taken in one single step. (This differs from traditional black pebbling games, where each step allows one single legal move.) The player wins the game if the sink node has been pebbled. The *cumulative complexity* (CC) of the pebbling is defined as the cumulative sum of the number of pebbles on the graph in each step, and the CC of the graph $\mathsf{cc}(\mathbb{G})$ is the minimal CC over all pebbling strategies.

Intuitively, a pebbling strategy is equivalent to an evaluation strategy for $\mathcal{F}_{\mathbb{G},\mathcal{H}}$ which only remembers labels in its memory. In [7] it was shown that such strategies are essentially optimal, i.e., $\mathsf{CMC}(\mathcal{F}_{\mathbb{G},\mathcal{H}}) \approx \mathsf{cc}(\mathbb{G}) \cdot W$.

<u>Depth-robust graphs.</u> *Depth-robust graphs* are class of graphs with high CC: Specifically, we say a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ is (e, d)-depth-robust if, after removing any nodes set $S \subseteq \mathbb{V}$ where $|S| \leq e$, the subgraph $\mathbb{G} - S$ still has a path with length d. Previous work [3] proved that any (e, d)-depth-robust graph $\mathbb{G}$ has cumulative complexity $\mathsf{cc}(\mathbb{G}) \geq e \cdot d$. Also, they show that constructions of (constant indegree) depth-robust graphs with $de = \Omega(N^2/\log(N))$ exist, which gives best-possible CC [1]. Later on, Alwen, Blocki, and Harsha [2] gave a procedure that samples with high probability a graph with the same depth-robustness guarantees, with a much simpler description.

<u>Our contributions.</u> Our two main contributions provide generic methods to devise constructions of MHFs from a simple primitive, like a block cipher, a permutation (that can be instantiated from a *fixed-key* block cipher), or a compression function. We consider a natural extension of the above model where the adversary queries an ideal version of the primitive.

1. We first define a simple class of so-called *small-block* labeling functions $\mathcal{H}_{\mathsf{fix}}$ which make one call to an underlying primitive. They transform a hard-to-pebble graph $\mathbb{G}$ (with maximal indegree 2) into a memory hard function $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ as described above, but where $\mathcal{H}$ is now instantiated from the underlying primitive via $\mathcal{H}_{\mathsf{fix}}$, and the resulting block length is L, the output length of the primitive. (E.g., if we used AES, then $L = 128$ bits.) Moreover, we prove that the CMC of $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ is approximately $\mathsf{cc}(\mathbb{G}) \cdot L$.
2. We then consider the problem of extending the block length of an iMHF to *increase memory hardness without changing* the underlying graph $\mathbb{G}$. To this end, from $\mathcal{H}_{\mathsf{fix}}$ with output length L, we define and construct a class of *wide-block* labeling functions, which in fact support *variable* block length. For any tunable parameters $\delta, W = 2^w \in \mathbb{N}$, the wide-block hash function

$\mathcal{H}_{\delta,w} : \{0,1\}^{\delta W} \to \{0,1\}^{W}$ turns any depth-robust graph $\mathbb{G}$ (with maximal indegree δ) into a memory hard function $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ via graph labeling. The CMC of $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ is approximately

$$\mathsf{CMC}(\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}) \approx \mathsf{cc}(\mathbb{G}) \cdot \delta W^3/L^2 \ .$$

Note that this is larger than $\mathsf{cc}(\mathbb{G}) \cdot W$ because, intuitively, we need to make multiple calls to the primitive to evaluate H, and use extra memory. In particular, we prove that the evaluation of $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ can be done sequentially with time $N\delta(W/L)^2$ and memory $N \cdot W$, i.e., the resulting CMC is $N^2\delta W^3/L^2$, via a naive strategy which runs N times the best-possible algorithm to evaluate $\mathcal{H}$, and keeps all W-bit labels in memory, in addition to internal states. Hence, if $\mathsf{cc}(\mathbb{G}) = \Theta(N^2/\log N)$ and $\delta = O(1)$, this means that the best possible CMC can have a gain of a factor at most $O(\log N)$ over the "naive" sequential strategy. This is the same upper bound on the speed-up we could establish for iMHFs from monolithic random oracles.[2]

We stress that because these results are generic, constructions can be obtained by using any graph $\mathbb{G}$ (or distribution over graphs) with sufficient depth-robustness guarantees.[3] We give some more details about these results next.

Small-Block Labeling Function: Constructions and Intuition. The small-block labeling functions $\mathcal{H}_{\mathsf{fix}}$ takes an input[4] $x \in \{0,1\}^L \cup \{0,1\}^{2L}$ and outputs an L-bit label. For a compression function cf, the resulting output is $\mathsf{cf}(x)$; for an ideal cipher ic, we split the input into a key part $k \in \{0,1\}^L \cup \{\perp\}$ (where $\perp$ is a designated key separate from the L-bit strings, which as with compression functions, will be necessary to implement variable input length) and an input part $x \in \{0,1\}^L$, the resulting output is $\mathsf{ic}(k,x) \oplus x$; for a random permutation rp, we denote as x^* the exclusive-or sum of L-bit input blocks and the output is $\mathsf{rp}(x^*) \oplus x^*$.

For any graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, the memory hardness of $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ is argued similarly as in previous work [7]. The high level idea is to transform the execution of any algorithm A that computes the MHF into an *ex-post-facto* pebbling for the graph $\mathbb{G}$, and argue that the cumulative memory complexity of A is proportional to the cumulative complexity of the pebbling. Here we generalize the technique of [7] – which relies on a compression argument – so that it works even if:

1. The ideal-primitive input contains no explicit information of the node v. (This was not the case in prior work.)

[2] Actually for certain graphs, we prove that the efficiency gap can be reduced to the optimal bound $O(1)$, by giving a more memory-efficient *sequential* algorithm.

[3] Our first result in fact only requires a lower bound on $\mathsf{cc}(\mathbb{G})$. It is an interesting open question to provide a wide-block labeling function which only relies on a lower bound for $\mathsf{cc}(\mathbb{G})$, rather than the (stronger) depth-robustness requirement.

[4] We assume the compression function allows both L- and $2L$-bit inputs, though most compression functions do not allow this by design. This could however be easily achieved by reserving one bit of the input to implement domain separation, and then padding short inputs.

2. The adversary can make *inverse* queries to the ideal primitive (as we also consider block ciphers now).
3. The input length of the primitive is fixed, and usually shorter than the actual input length of the labeling function.

Remark 1. Note that Blocki et al. [11] independently proposed a proof that addressed the first and the third challenge. But to the best of our knowledge, our technique is the only one that works even if the adversary makes *inverse* queries.

<u>Succinct MHFs from Wide-Block Labeling Functions.</u> Given a small-block labeling function $\mathcal{H}_{\mathsf{fix}}$ and tunable parameters $\delta, W = 2^w \in \mathbb{N}$, the wide-block labeling function $\mathcal{H}_{\delta,w} : \{0,1\}^{\delta W} \to \{0,1\}^{W}$ is essentially a graph labeling function built upon $\mathcal{H}_{\mathsf{fix}}$ and a gadget graph $\mathbb{G}_{\delta,W}$. $\mathbb{G}_{\delta,W}$ is the *composition* of two subgraphs, namely, a MIX graph $\mathbb{G}_{\mathsf{mix}}$, and a *source-to-sink depth robust graph* $\mathbb{G}_{\mathsf{ssdr}} = (\mathbb{V}^{'}, \mathbb{E}^{'})$ that satisfies for any subset $S \subseteq \mathbb{V}^{'}$ (with bounded size), $\mathbb{G}_{\mathsf{ssdr}} - S$ has a long path starting from a source node of $\mathbb{G}_{\mathsf{ssdr}}$ and ending at a sink node of $\mathbb{G}_{\mathsf{ssdr}}$.

For any (e, d)-depth-robust graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, the CMC of $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ is argued by opening the graph structure underlying $\mathcal{H}_{\delta,w}$, and consider $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ as a graph function built upon $\mathcal{H}_{\mathsf{fix}}$ and *a bootstrapped graph* $\mathsf{Ext}_{\delta,W}(\mathbb{G})$. We show that the graph $\mathsf{Ext}_{\delta,W}(\mathbb{G})$ is extremely depth-robust and thus has pebbling complexity $\Omega(\delta W^3/L^3) \cdot ed$. Then by the property of $\mathcal{H}_{\mathsf{fix}}$, we can build the connection between the CMC of $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ and the CC of $\mathsf{Ext}_{\delta,W}(\mathbb{G})$.

<u>Depth-robustness of $\mathsf{Ext}_{\delta,W}(\mathbb{G})$.</u> Given any (bounded-size) nodes subset S of $\mathsf{Ext}_{\delta,W}(\mathbb{G})$, we show the existence of an extremely long path in $\mathsf{Ext}_{\delta,W}(\mathbb{G}) - S$ in three steps: First, S is transformed into a small set $S^{'}$ in $\mathbb{G}$, and we obtain a long path P in $\mathbb{G} - S^{'}$ by depth-robustness of $\mathbb{G}$; second, by *source-to-sink depth-robustness* of the SSDR graph, each vertex v in P is transformed into a path P_v^* in $\mathsf{Ext}_{\delta,W}(\mathbb{G}) - S$; finally, the structure of the MIX graph helps to elegantly connect the paths in $\{P_v^*\}_{v \in P}$ into an extremely long path in $\mathsf{Ext}_{\delta,W}(\mathbb{G}) - S$.

Remark 2. Note that our wide-block labeling functions can only turn *depth-robust graphs* (instead of arbitrary graphs with high CC) into memory hard functions. It is hard to link CMC and $\mathsf{cc}(\mathbb{G})$ directly using our extension framework. The hardness lies in linking $\mathsf{cc}(\mathsf{Ext}_{\delta,W}(\mathbb{G}))$ and $\mathsf{cc}(\mathbb{G})$. In particular, even if the gadget graph $\mathbb{G}_{\delta,W}$ has high CC, we do not know how to prove that $\mathsf{cc}(\mathsf{Ext}_{\delta,W}(\mathbb{G})) \geq \mathsf{cc}(\mathbb{G}) \cdot \mathsf{cc}(\mathbb{G}_{\delta,W})$. This is because we do not know how to transform a pebbling P_1 (of $\mathsf{Ext}_{\delta,W}(\mathbb{G})$) into a *legal* pebbling P_2 (of $\mathbb{G}$), and argue that $\mathsf{cc}(\mathsf{P}_1)$ is *at least* $\mathsf{cc}(\mathsf{P}_2)$ times $\mathsf{cc}(\mathbb{G}_{\delta,W})$.

2 Preliminaries

<u>Notation.</u> Let $\mathbb{N}$ and $\mathbb{R}$ denote the sets of natural numbers and real numbers respectively. Denote by $[n]$ the set of integers $\{1, \ldots, n\}$. By $\log(\cdot)$ we always

refer to binary logarithm. For strings x and y, $|x|$ is the length of x and we use $x \| y$ or (x, y) to denote the concatenation of x and y. For a set $\mathbb{X}$, $x \xleftarrow{\$} \mathbb{X}$ is the process of assigning to x an element picked uniformly from $\mathbb{X}$, and $|\mathbb{X}|$ denotes the number of elements in $\mathbb{X}$. For a distribution $\mathcal{D}$, we use $x \leftarrow \mathcal{D}$ to denote the sampling of x from distribution $\mathcal{D}$. For an algorithm A, we use $\mathsf{A}(x; r)$ to denote the output of the algorithm on input x and random coins r.

2.1 Memory-Hard Functions in the Parallel Ideal Primitive Model

Ideal Primitives. In this paper, we consider three ideal primitives, namely, *compression functions*, *ideal ciphers*, and *random permutations*. All primitives will have an understood block length $L = 2^{\ell}$, which is assumed to be a power of two. In the following context, we will omit L in the ideal-primitive notation if there is no ambiguity.

Denote by $\mathbb{CF}$ the set of functions[5] with domain $\{0,1\}^L \cup \{0,1\}^{2L}$ and image $\{0,1\}^L$. Denote by $\mathbb{IC}$ the set of keyed permutations with domain $\mathcal{K} \times \{0,1\}^L$ and image $\{0,1\}^L$. For simplicity, the key space is set as $\mathcal{K} := \{\perp\} \cup \{0,1\}^L$ in the following context. Finally, we let $\mathbb{RP}$ be the set of permutations with input/output length L.

Parallel Ideal Primitive Model. Towards modeling the computation of memory-hard functions, we generalize the *Parallel Random Oracle Model* defined by Alwen and Serbinenko [7] to *Parallel Ideal Primitive Model*. Let $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ be a type of ideal primitive set. For any oracle-aided algorithm A, input x and internal randomness r, the execution $\mathsf{A}(x; r)$ works as follows. First, a function ip (with block length L) is uniformly chosen from the set $\mathbb{IP}$. The oracle-aided algorithm A can make oracle query to ip as follows: If $\mathsf{ip} = \mathsf{cf}$ is a randomly sampled compression function, the algorithm can make queries with form $(\text{“}\mathbb{CF}\text{”}, x)$ and receive value $\mathsf{cf}(x)$. If $\mathsf{ip} = \mathsf{ic}$ is a randomly sampled ideal cipher, the algorithm can make *forward* queries with form $(\text{“}\mathbb{IC}\text{”}, +, k, x)$ and receive value $\mathsf{ic}(k, x)$, or make *inverse* queries with form $(\text{“}\mathbb{IC}\text{”}, -, k, y)$ and receive value $\mathsf{ic}^{-1}(k, y)$. Similarly, if $\mathsf{ip} = \mathsf{rp}$ is a randomly sampled permutation, the algorithm can make *forward* queries with form $(\text{“}\mathbb{RP}\text{”}, +, x)$ and receive value $\mathsf{rp}(x)$, or make *inverse* queries with form $(\text{“}\mathbb{RP}\text{”}, -, y)$ and receive value $\mathsf{rp}^{-1}(y)$.

Let $\sigma_0 = (x, \emptyset)$ denote the initial input state. For each round $i \in \mathbb{N}$, $\mathsf{A}(x; r)$ takes input state σ_{i-1}, performs *unbounded* computation, and generates an output state $\bar{\sigma}_i = (\delta_i, \mathbf{q}_i, \mathbf{out}_i)$, where δ_i is a binary string, $\mathbf{q}_i$ is a vector of queries to the ideal primitive ip, and each element of $\mathbf{out}_i$ is with form (v, ℓ_v) where v and ℓ_v are L-bit output labels. We define $\sigma_i = (\delta_i, \mathbf{ans}(\mathbf{q}_i))$ to be the input state for round $i + 1$, where $\mathbf{ans}(\mathbf{q}_i)$ is the vector of ideal primitive answers to $\mathbf{q}_i$. The execution terminates after round $T \in \mathbb{N}$ if $|\mathbf{q}_T| = 0$. We use $\mathsf{A}^{\mathsf{ip}}(x; r)$

[5] We assume the compression function allows both L- and $2L$-bit inputs, though most compression functions do not allow this by design. This could however be easily achieved by reserving one bit of the input to implement domain separation, and then padding short inputs.

to indicate both the execution output (i.e., the concatenation of output labels) and the execution *trace* (i.e., all of the input and output states $(\sigma_0, \bar{\sigma}_1, \sigma_1, \dots)$). We also assume an upper bound q on the total number of output labels/ideal primitive queries that an algorithm makes, i.e., $\sum_{i \geq 1} |\mathbf{q}_i| + |\mathbf{out}_i| \leq \mathsf{q}$. We call A a *sequential* algorithm if $|\mathbf{q}_i| = 1$ for every $1 \leq i < T$, otherwise A is a *parallel* algorithm.

COMPLEXITY MEASURES. Given the trace $\mathsf{A}^{\mathsf{ip}}(x;r)$ on input x, randomness r and ideal primitive ip, we define as time complexity $\mathsf{Tm}(\mathsf{A}^{\mathsf{ip}}(x;r))$ the number of rounds ran by A, and space complexity $\mathsf{Spc}(\mathsf{A}^{\mathsf{ip}}(x;r))$ the size of the maximal input state. Moreover, we define $\mathsf{Tm}(\mathsf{A})$ (and $\mathsf{Spc}(\mathsf{A})$) to be the maximal time (and space) complexity of A over all choices of x, r and ip.[6] We define cumulative memory complexity (CMC) [7] in the parallel ideal primitive model.

Definition 1 (Cumulative Memory Complexity). *Given trace* $\mathsf{A}^{\mathsf{ip}}(x;r)$ *(with input states* $(\sigma_0, \sigma_1, \dots)$*). we define cumulative memory complexity*

$$\mathsf{CMC}(\mathsf{A}^{\mathsf{ip}}(x;r)) := \sum_{i=0}^{\mathsf{Tm}(\mathsf{A}^{\mathsf{ip}}(x;r))} |\sigma_i|$$

to be the sum of input states' size over time. For a real value $\epsilon \in [0,1]$*, and a family of functions* $\mathcal{F} = \{f^{\mathsf{ip}} : \mathcal{X} \to \mathcal{Y}\}_{\mathsf{ip} \in \mathbb{IP}}$*, we define the* ϵ*-cumulative memory complexity of* $\mathcal{F}$ *to be*

$$\mathsf{CMC}_\epsilon(\mathcal{F}) := \min_{x \in \mathcal{X}, \mathsf{A} \in \mathcal{A}_{x,\epsilon}} \mathbb{E}[\mathsf{CMC}(\mathsf{A}^{\mathsf{ip}}(x;r))],$$

where the expectation is taken over the uniform choices of ip *and* r*.* $\mathcal{A}_{x,\epsilon}$ *is the set of parallel algorithms*[7] *that satisfy the following: with probability at least* ϵ *(over the uniform choices of* ip *and* r*), the algorithm on input* x *and oracle* ip *outputs* $f^{\mathsf{ip}}(x)$*.*

MEMORY-HARD FUNCTIONS. We now define memory hard functions in the parallel ideal primitive model. Intuitively, there exists a relatively efficient *sequential* algorithm that computes the MHFs, and any *parallel* algorithm that evaluates the functions incurs high CMC cost.

Definition 2 (Memory Hard Functions). *For an ideal primitive set* $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$*, a family of functions* $\mathcal{F} = \{f^{\mathsf{ip}} : \mathcal{X} \to \mathcal{Y}\}_{\mathsf{ip} \in \mathbb{IP}}$ *is* $(\mathsf{C}^{\parallel}_{\mathcal{F}}, \Delta_{\mathcal{F}}, T_{\mathcal{F}})$*-memory hard if and only if the following properties hold. (*$\mathsf{C}^{\parallel}_{\mathcal{F}}, \Delta_{\mathcal{F}}$ *are functions that take as input a real value in* $(0,1]$ *and output a real value,* $T_{\mathcal{F}}$ *is an integer.)*

[6] $\mathsf{Tm}(\mathsf{A})$ (and $\mathsf{Spc}(\mathsf{A})$) measure *worst-case sequential efficiency* by providing *upper bounds* on time/memory.

[7] Recall that the total number of output labels/ideal primitive queries that the algorithm makes is at most q.

Memory-hardness: *For any* $\epsilon \in (0,1]$, *we have* $\mathsf{CMC}_\epsilon(\mathcal{F}) \geq \mathsf{C}^{\parallel}_{\mathcal{F}}(\epsilon)$.
Efficiency-gap: *For any* $\epsilon \in (0,1]$, *it holds that*

$$\frac{\min_{\mathsf{A}\in\mathcal{A}_{\mathcal{F},T_{\mathcal{F}}}}(\epsilon \cdot \mathsf{Tm}(\mathsf{A}) \cdot \mathsf{Spc}(\mathsf{A}))}{\mathsf{CMC}_\epsilon(\mathcal{F})} \leq \Delta_{\mathcal{F}}(\epsilon),$$

where $\mathcal{A}_{\mathcal{F},T_{\mathcal{F}}}$ *is the set of deterministic sequential algorithms that run in at most* $T_{\mathcal{F}}$ *steps and correctly output* $f^{\mathsf{ip}}(x)$ *for any* ip *and* x.

2.2 Graphs and Pebbling Models

<u>Graph Notations.</u> We use $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ to denote a directed acyclic graph (DAG) with $N = 2^n$ nodes, where $\mathbb{V} = \{1, \ldots, N\}$. Let $\mathsf{src}(\mathbb{G}) \subseteq \mathbb{V}$ be the set of source nodes and $\mathsf{sink}(\mathbb{G}) \subseteq \mathbb{V}$ be the set of sink nodes. For a node $v \in \mathbb{V}$, $\mathsf{pred}(v) := \{u : (u,v) \in \mathbb{E}\}$ are the predecessor nodes of v, $\mathsf{succ}(v) := \{w : (v,w) \in \mathbb{E}\}$ is the set of v's successors. We use $\mathsf{indeg}(v) := |\mathsf{pred}(v)|$ to denote the indegree of v, and $\mathsf{indeg}(\mathbb{G}) := \max_{v\in\mathbb{V}} \mathsf{indeg}(v)$ is the indegree of $\mathbb{G}$. For a directed acyclic path P, the length of P is the number of nodes it traverses. $\mathsf{depth}(\mathbb{G})$ is the length of the longest path in $\mathbb{G}$. For a nodes set $S \subseteq \mathbb{V}$, $\mathbb{G} - S$ is the DAG obtained from $\mathbb{G}$ by removing S and incident edges. Next, we review a useful graph-theoretic property called depth-robustness.

Definition 3 (Depth-Robustness [3]**).** *A DAG* $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ *is* (e,d)*-depth-robust if and only if* $\mathsf{depth}(\mathbb{G} - S) \geq d$ *for any* $S \subseteq \mathbb{V}$ *where* $|S| \leq e$.

Next, we define a stronger notion of depth-robustness called *source-to-sink-depth-robustness*. Intuitively, it means that after removing any nodes set with certain size, there still exists a long path from a source node to a sink node.

Definition 4 (Source-to-Sink-Depth-Robustness). *A DAG* $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ *is* (e,d)*-source-to-sink-depth-robust if and only if for any* $S \subseteq \mathbb{V}$ *where* $|S| \leq e$, $\mathbb{G} - S$ *has a path (with length at least* d*) that starts from a source node of* $\mathbb{G}$ *and ends up in a sink node of* $\mathbb{G}$.

<u>Graph Pebbling.</u> We consider a pebbling game played on a DAG $(\mathbb{G} = \mathbb{V}, \mathbb{E})$ [7]. We denote by a *parallel* pebbling on $\mathbb{G}$ as a sequence of pebbling configurations $\mathsf{P} = (\mathsf{P}_0, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$ where $\mathsf{P}_0 = \emptyset$ and $\mathsf{P}_i \subseteq \mathbb{V}$ $(1 \leq i \leq t_{\mathsf{peb}})$. We define two properties for P.

- *Legality:* We say P is legal if it satisfies follows: A pebble can be put on a node $v \in \mathbb{V}$ only if v is a *source* node or v's predecessors were all pebbled at the end of the previous step, that is, for any $i \in [t_{\mathsf{peb}}]$ and any $v \in \mathsf{P}_i \setminus \mathsf{P}_{i-1}$, it holds that $\mathsf{pred}(v) \subseteq \mathsf{P}_{i-1}$.[8]
- *Successfulness:* We say P is successful if it satisfies follows: Every sink node has been pebbled at least once, that is, for any $v \in \mathsf{sink}(\mathbb{G})$, there exists $i \in [t_{\mathsf{peb}}]$ such that $v \in \mathsf{P}_i$.

[8] $\mathsf{pred}(v) = \emptyset$ for $v \in \mathsf{src}(\mathbb{G})$.

We say P is a *sequential* pebbling if it further satisfies that $|\mathsf{P}_i \setminus \mathsf{P}_{i-1}| = 1$ for every $i \in [t_{\mathsf{peb}}]$. Next we define pebbling complexities of graphs.

Definition 5 (Complexity Measures [7]). *For a pebbling strategy* $\mathsf{P} = (\mathsf{P}_0 = \emptyset, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$*, we define the cumulative complexity (and ST-complexity) of* P *to be*

$$\mathsf{cc}(\mathsf{P}) := \sum_{i=0}^{t_{\mathsf{peb}}} |\mathsf{P}_i|, \qquad \mathsf{st}(\mathsf{P}) := t_{\mathsf{peb}} \cdot \max_{i \in [t_{\mathsf{peb}}]} (|\mathsf{P}_i|).$$

For a DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$*, let* $\mathcal{P}^{\|}(\mathbb{G})$ *be the set of parallel pebblings of* $\mathbb{G}$ *(that are legal and successful); for any* $t \in \mathbb{N}$*, let* $\mathcal{P}_t(\mathbb{G})$ *be the set of sequential pebblings of* $\mathbb{G}$ *that (are legal and successful) and* takes at most t steps. *We define the cumulative complexity (and ST-complexity) of* $\mathbb{G}$ *to be*

$$\mathsf{cc}(\mathbb{G}) := \min_{\mathsf{P} \in \mathcal{P}^{\|}(\mathbb{G})} \mathsf{cc}(\mathsf{P}), \qquad \mathsf{st}(\mathbb{G}, t) := \min_{\mathsf{P} \in \mathcal{P}_t(\mathbb{G})} \mathsf{st}(\mathsf{P}).$$

There is a tight relation between depth-robustness and cumulative complexity.

Lemma 1 ([3]). *Let* $\mathbb{G}$ *be an* (e, d)*-depth-robust DAG, then* $\mathsf{cc}(\mathbb{G}) \geq e \cdot d$.

2.3 Graph-Based iMHFs from Labeling Functions

For a DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with $N = 2^n$ nodes, we index the nodes set $\mathbb{V} = \{1, \ldots, N\}$ by a topological order so that $\mathsf{src}(\mathbb{G}) = \{1, \ldots, n_s\}$ and $\mathsf{sink}(\mathbb{G}) = \{N - n_t + 1, \ldots, N\}$. Fix $W = 2^w$, $\delta := \mathsf{indeg}(\mathbb{G})$ and let $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ be the set of ideal primitives. Denote by

$$\mathcal{H} = \mathcal{H}_{\delta,w} = \left\{ \mathsf{lab}_\gamma^{\mathsf{ip}} : \{0,1\}^{\gamma W} \to \{0,1\}^W \right\}_{\gamma \in [\delta], \mathsf{ip} \in \mathbb{IP}}$$

a family of labeling functions. We define a family of *graph functions* $\mathcal{F}_{\mathbb{G},\mathcal{H}} = \{\mathsf{F}_{\mathbb{G},\mathcal{H}}^{\mathsf{ip}} : \{0,1\}^{n_s W} \to \{0,1\}^{n_t W}\}_{\mathsf{ip} \in \mathbb{IP}}$ based on $\mathbb{G}$ and $\mathcal{H}$: For an input $\mathbf{x} = (x_1, \ldots, x_{n_s}) \in \{0,1\}^{n_s W}$, denote $\ell_i := \mathsf{lab}_1^{\mathsf{ip}}(x_i)$ as the label of the ith source $(1 \leq i \leq n_s)$, we recursively define the label of $v \in [N]$ as

$$\ell_v := \mathsf{lab}_\gamma^{\mathsf{ip}}(\ell_{v_1}, \ldots, \ell_{v_\gamma})$$

where $(v_1, \ldots, v_\gamma)$ are the predecessors of v. The output $\mathsf{F}_{\mathbb{G},\mathcal{H}}^{\mathsf{ip}}(x)$ is defined as the concatenation of sinks' labels, that is, $(\ell_{N-n_t+1} \| \cdots \| \ell_N)$.

<u>Preprocessing the Inputs.</u> If $n_s = |\mathsf{src}(\mathbb{G})| > 1$, we implicitly assume that the input vector $\mathbf{x} = (x_1, \ldots, x_{n_s})$ has no overlapping blocks (and we call it a *non-colliding* input vector), that is, for any $i, j \in \mathbb{N}$ such that $i \neq j$, we have $x_i \neq x_j$. This constraint is necessary for preventing the adversary from easily saving memory. For example, if the blocks in the input vector are identical, the adversary only needs to store a single block instead of the entire input vector. The constraint/assumption is also reasonable as we can re-randomize the original input using a random oracle $\mathsf{RO} : \{0,1\}^{n_s W} \to \{0,1\}^{n_s W}$, and the output blocks will be distinct with overwhelming probability when $n_s \ll 2^{W/2}$.

Remark 3 (Graph constraint). In the following context, the graphs we are concerned with should satisfy certain properties. In particular, each 2-indegree DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ considered in Sect. 3 should be *predecessors-distinct*, that is, for any two distinct vertices $u, v \in \mathbb{V} \backslash \mathsf{src}(\mathbb{G})$, we have $\mathsf{pred}(u) \neq \mathsf{pred}(v)$. Looking ahead, this constraint is used to prevent non-source nodes label collisions. Additionally, each δ-indegree DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ considered in Sect. 4 satisfies a property called *first-predecessor-distinctness*, that is, there exists a map from each non-source node $v \in \mathbb{V} \backslash \mathsf{src}(\mathbb{G})$ to a single node $\mathsf{fpre}(v) \in \mathsf{pred}(v)$, so that for any two distinct vertices $u, v \in \mathbb{V} \setminus \mathsf{src}(\mathbb{G})$, we have $\mathsf{fpre}(u) \neq \mathsf{fpre}(v)$. Looking ahead, this constraint is used to guarantee that the 2-indegree bootstrapped graph $\mathsf{Ext}_{\delta,W}(\mathbb{G})$ built upon $\mathbb{G}$ is *predecessors-distinct*. We stress that practical DAG constructions usually contain a subpath that traverses all of the vertices, and thus are both first-predecessor-distinct and predecessors-distinct.[9]

3 MHFs from Small-Block Labeling Functions

In this section, we construct a family of graph-based iMHFs based on what we refer to as a *small-block labeling function*, i.e., a simple hash function based on an ideal primitive, which preserves its block length. (Note that this notion is introduced for modularity reason – we could define our designs directly as depending on a primitive.) In Sect. 3.1, we define and construct of *efficient* small-block labeling functions from primitives, and in Sect. 3.2, we prove that the constructions satisfies the required properties. Finally, in Sect. 3.3, we construct iMHFs from small-block labeling functions.

3.1 Small-Block Labeling Functions: Definition and Construction

Definition 6 (Small-Block Labeling Functions). *For an ideal primitive $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ with block length $L = 2^\ell$, we say*

$$\mathcal{H}_{\mathsf{fix}} = \left\{ \mathsf{flab}^{\mathsf{ip}} : \left\{ \{0,1\}^L \cup \{0,1\}^{2L} \right\} \to \{0,1\}^L \right\}_{\mathsf{ip} \in \mathbb{IP}}$$

is a family of β-small-block labeling functions if it has the following property.

> $\beta(\cdot,\cdot)$**-pebbling reducibility:** *For any $\epsilon \in (0, 1]$ and 2-indegree (predecessors-distinct[10]) DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$,[11] let $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ be the graph function built upon $\mathbb{G}$ and $\mathcal{H}_{\mathsf{fix}}$. We have*
>
> $$\mathsf{CMC}_\epsilon(\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}) \geq \beta(\epsilon, \log|\mathbb{V}|) \cdot \mathsf{cc}(\mathbb{G}),$$
>
> *where $\mathsf{cc}(\mathbb{G})$ is the cumulative complexity of $\mathbb{G}$ (Definition 5).*

[9] First-predecessor-distinctness holds as each non-source node v can pick her previous node in the subpath (that traverses all of the vertices) as their *first predecessor*; predecessors-distinctness holds as otherwise a cycle would exist.

[10] See Remark 3 for definition of predecessors-distinct graphs.

[11] $\mathbb{G}$ can have multiple source/sink nodes.

Construction. Next, we show how to construct small-block labeling functions from ideal primitives. Our main contribution is a construction from a random permutation, which can be instantiated from fixed-key AES. For completeness, we also present constructions from ideal ciphers and compression functions. We fix the input domain to be $\{0,1\}^L \cup \{0,1\}^{2L}$ and the output space to be $\{0,1\}^L$, and denote as $\mathbb{RP}$, $\mathbb{IC}$, $\mathbb{CF}$ the random permutations, ideal ciphers, and compression functions, respectively.

1. Given any $\mathsf{rp} \in \mathbb{RP}$, we define the labeling function $\mathsf{flab}^{\mathsf{rp}}(\cdot)$ as follows: For any input $x \in \{0,1\}^L$, the output is $\mathsf{flab}^{\mathsf{rp}}(x) := \mathsf{rp}(x) \oplus x$; for any input $(x_1, x_2) \in \{0,1\}^{2L}$, denote as $x^* := x_1 \oplus x_2 \in \{0,1\}^L$, the output is $\mathsf{flab}^{\mathsf{rp}}(x_1, x_2) := \mathsf{rp}(x^*) \oplus x^*$.
2. Given any $\mathsf{ic} \in \mathbb{IC}$, we define the labeling function $\mathsf{flab}^{\mathsf{ic}}(\cdot)$ as follows: For any input $x \in \{0,1\}^L$, the output is $\mathsf{flab}^{\mathsf{ic}}(x) := \mathsf{ic}(\perp, x) \oplus x$; for any input $(k, x) \in \{0,1\}^{2L}$, the output is $\mathsf{flab}^{\mathsf{ic}}(k, x) := \mathsf{ic}(k, x) \oplus x$.
3. Given any $\mathsf{cf} \in \mathbb{CF}$, we define the labeling function $\mathsf{flab}^{\mathsf{cf}}(\cdot)$ as follows: For any input $x \in \{0,1\}^L \cup \{0,1\}^{2L}$, the output is $\mathsf{flab}^{\mathsf{cf}}(x) := \mathsf{cf}(x)$.

Note that all of the above constructions are highly efficient as they call the ideal-primitive only once. Next we show that the constructions are pebbling reducible.

Remark 4. The construction for compression functions is interesting, even in view of prior work, because of the fact that prior work included the node identity into the hash-function input, thus effectively requiring $2L + \log N$-bit inputs, whereas here we can get away with $2L$.

3.2 Small-Block Labeling Functions: Pebbling Reducibility

In this section, we show that the labeling functions constructed in Sect. 3.1 satisfy pebbling reducibility, via the following three theorems.

Theorem 1. *Assume an adversary can make no more than* q_1 *oracle calls and* q_2 *output calls such that* $\mathsf{q}_1 + \mathsf{q}_2 = \mathsf{q} = 2^{L/4}$. $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{cf}}\}_{\mathsf{cf} \in \mathbb{CF}}$ *built upon compression function* $\mathbb{CF}$ *is* $\beta(\cdot,\cdot)$*-pebbling reducible, where for all* $\epsilon \geq 3 \cdot 2^{-L/8}$ *and* $N \leq 2^{L/8}$*, it holds that* $\beta(\epsilon, \log N) \geq \frac{\epsilon L}{8}$.

Theorem 2. *Assume an adversary can make no more than* q_1 *oracle calls and* q_2 *output calls such that* $\mathsf{q}_1 + \mathsf{q}_2 = \mathsf{q} = 2^{L/4}$. $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{ic}}\}_{\mathsf{ic} \in \mathbb{IC}}$ *built upon ideal cipher* $\mathbb{IC}$ *is* $\beta(\cdot,\cdot)$*-pebbling reducible, where for all* $\epsilon \geq 3 \cdot 2^{-L/8}$ *and* $N \leq 2^{L/8}$*, it holds that* $\beta(\epsilon, \log N) \geq \frac{\epsilon L}{8}$.

Theorem 3. *Assume an adversary can make no more than* q_1 *oracle calls and* q_2 *output calls such that* $\mathsf{q}_1 + \mathsf{q}_2 = \mathsf{q} = 2^{L/8}$. $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{rp}}\}_{\mathsf{rp} \in \mathbb{RP}}$ *built upon random permutation* $\mathbb{RP}$ *is* $\beta(\cdot,\cdot)$*-pebbling reducible, where for all* $\epsilon \geq 3 \cdot 2^{-L/10}$ *and* $N \leq 2^{L/10}$*, it holds that* $\beta(\epsilon, \log N) \geq \frac{\epsilon L}{40}$.

Next we present the proofs for Theorems 1, 2 and 3. First, we introduce some notation for graph labeling, then we highlight the proof techniques and introduce the notion of ex-post-facto pebbling in the ideal primitive model. Finally, we provide the formal proof.

GRAPH LABEL NOTATIONS. Fix ideal primitive $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$,[12] input vector $\mathbf{x}$ and any graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$. For a primitive $\mathsf{ip} \in \mathbb{IP}$ and any node $v \in \mathbb{V}$, we denote as ℓ_v the graph label of v. If v is a source, $\mathsf{prelab}(v)$ is the corresponding input label x_v. If v is a non-source node, we define $\mathsf{prelab}(v)$ based on the type of the ideal primitive:

– If $\mathbb{IP} = \mathbb{RP}$, we define $\mathsf{prelab}(v)$ as the exclusive-or sum of v's parents' ℓ-labels.
– If $\mathbb{IP} = \mathbb{IC}/\mathbb{CF}$, we define $\mathsf{prelab}(v)$ as the concatenation of v's parents' ℓ-labels.

Similarly, for every node $v \in \mathbb{V}$, we define $\mathsf{aftlab}(v)$ based on the type of the ideal primitive:

– If $\mathbb{IP} = \mathbb{RP}/\mathbb{CF}$, we define $\mathsf{aftlab}(v) = \mathsf{ip}(\mathsf{prelab}(v))$.
– If $\mathbb{IP} = \mathbb{IC}$ and v has only one parent, we define $\mathsf{aftlab}(v) = \mathsf{ip}(\perp, \mathsf{prelab}(v))$. Otherwise if v has two parents, denote as $\mathsf{prelab}(v) = (y_1, y_2)$ (where $y_1, y_2 \in \{0,1\}^L$ are ℓ-labels of v's parents), we define $\mathsf{aftlab}(v)$ as $(y_1, \mathsf{ip}(y_1, y_2))$.

In the following context, we abuse the notation a bit in that if $\mathsf{prelab}(v)$ (or $\mathsf{aftlab}(v)$) is an L-bit string and ip is an ideal cipher, we use $\mathsf{ip}(\mathsf{prelab}(v))$ (or $\mathsf{ip}^{-1}(\mathsf{aftlab}(v))$) to denote $\mathsf{ip}(\perp, \mathsf{prelab}(v))$ (or $\mathsf{ip}^{-1}(\perp, \mathsf{aftlab}(v))$). Moreover, for an ideal cipher query with input $x_c = (\perp, x)$, we say $x_c = \mathsf{prelab}(v)$ (or $x_c = \mathsf{aftlab}(v)$) if and only if $x = \mathsf{prelab}(v)$ (or $x = \mathsf{aftlab}(v)$), respectively.

We remark that $\mathsf{prelab}(v)$ (and $\mathsf{aftlab}(v)$) are more than just single labels, they are used to identify the node from a query input to the ideal primitive.

PROOF HIGHLIGHT. Similar as in [7], the proof idea is to transform any algorithm execution A into an *ex-post-facto* pebbling, and argue that the cumulative memory complexity of A is proportional to the cumulative complexity of the pebbling. This is proved by mapping each node $v \in \mathbb{V}$ to an ideal-primitive entry $(\mathsf{prelab}(v), \mathsf{ip}(\mathsf{prelab}(v)))$, and argue that for each round $i \in \mathbb{N}$, the input state σ_i should have large size as it is an encoding for many ideal-primitive entries. In particular, for every node v in the ith pebbling configuration, $\mathsf{ip}(\mathsf{prelab}(v))$ (and ℓ_v) can be decoded from an oracle-call input in the partial execution $\mathsf{A}(\sigma_i)$. Here we generalize the technique of [7] so that it works even if:

1. The ideal-primitive input $\mathsf{prelab}(v)$ contains no explicit information of the node index v. (Note that in previous work [7], $\mathsf{prelab}(v)$ has v as a prefix.)
2. The adversary can make *inverse* queries to the ideal primitive.
3. The input length of the primitive is much shorter than the actual input length of the labeling function.

[12] $\mathbb{CF}$ denotes the compression function, $\mathbb{IC}$ denotes the ideal cipher, and $\mathbb{RP}$ denotes the random permutation.

Ex-post-facto Pebbling. Similar as in [7], we define a notion called *ex-post-facto pebbling in the ideal primitive model.* Fix ideal primitive $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$, input vector $\mathbf{x}$, DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$. For any $\mathsf{ip} \in \mathbb{IP}$, randomness r, and the execution trace $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ (that runs for $t_{\mathsf{peb}} + 1$ rounds), we turn the trace into an *ex-post-facto pebbling*

$$\mathcal{P}(\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)) = (\mathsf{P}_0 = \emptyset, \ldots, \mathsf{P}_{t_{\mathsf{peb}}}).$$

For each oracle call/query that asks for the ideal-primitive value on input x_c, we say that the call is a *correct* call for a node $v \in \mathbb{V}$ if and only if x_c matches $\mathsf{prelab}(v)$ and the call is forward, or x_c matches $\mathsf{aftlab}(v)$ and the call is an inverse call. We define the ex-post-facto pebbling configurations in reverse order. For i from t_{peb} to 1, denote as σ_i the input state of round $i+1$ and $\mathsf{A}(\sigma_i)$ the partial execution of $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ after round i, the pebbling configuration P_i is defined as follows. (In the following context, by round γ, we always mean the γ-th round in the execution $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$.)

1. *Critical Calls:* We sort the output/ideal primitive calls of $\mathsf{A}(\sigma_i)$ in chronological order[13] and determine whether they are *critical* calls.
   - An output call (in round $\gamma > i$) with label (v, ℓ_v) is a *critical* call for $v \in \mathbb{V}$ if and only if v is a sink and in the trace $\mathsf{A}(\sigma_i)$, no correct call for v appeared before round γ.
   - An ideal-primitive call (in round $\gamma > i$) is a *critical* call for a node $u \in \mathbb{V}$ if and only if the following conditions both hold: (i) the ideal-primitive call is a correct call for a successor node $v \in \mathsf{succ}(u)$ and in the trace $\mathsf{A}(\sigma_i)$, no correct call for u appeared before round γ; (ii) v is in P_γ.
2. *Pebbling Configuration:* A node $v \in \mathbb{V}$ is included into the pebbling configuration P_i if and only if *both* of the following conditions hold:
   - There is at least one critical call for v in the trace $\mathsf{A}(\sigma_i)$.
   - There is at least one correct call for v between round 1 and round i (inclusively).[14]

In a critical call, the algorithm provides the information of a graph label without recomputing, hence the call is useful in extracting ideal-primitive entries. On a side note, by definition of critical call and ex-post-facto pebbling, for any round i, we might possibly put a node u into P_i only if u is a sink or one of its successor $v \in \mathsf{succ}(u)$ is in P_γ for some $\gamma > i$.

Proof (of Theorems 1, 2 and 3). Fix ideal primitive type $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$, input vector $\mathbf{x}$, adversary A, and any graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, the proof consists of the following four steps.

Label Collisions. First, we show that with probability at least $1 - \epsilon_{\mathsf{coll}}(\mathbb{IP})$ (over the choice of the ideal primitive), the pre-labels $\{\mathsf{prelab}(v)\}_{v \in \mathbb{V}}$ are all distinct.[15]

[13] We assume an implicit order for the calls in the same round.

[14] Note that the existence of a correct call for v in round i does not imply $v \in \mathsf{P}_i$, because v may not have a critical call in the future.

[15] If $\mathbb{IP} = \mathbb{IC}/\mathbb{RP}$, this also implies that $\{\mathsf{aftlab}(v)\}_{v \in \mathbb{V}}$ are distinct.

Lemma 2. *Fix ideal primitive* $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ *(with block length* L*), predecessors-distinct DAG* $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ *(with* n_s *source nodes), non-colliding input vector* $\mathbf{x} \in \{0,1\}^{n_s L}$,[16] *and* $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{ip}}\}_{\mathsf{ip} \in \mathbb{IP}}$ *constructed in Sect. 3. With probability at least* $1 - \epsilon_{\mathsf{coll}}(\mathbb{IP})$ *(over the uniformly random choice of* ip*), the graph labeling satisfies that the pre-labels* $\{\mathsf{prelab}(v)\}_{v \in \mathbb{V}}$ *are all distinct. Here* $\epsilon_{\mathsf{coll}}(\mathbb{CF}) = |\mathbb{V}|^2/2^{L+1}$ *and* $\epsilon_{\mathsf{coll}}(\mathbb{IC}) = \epsilon_{\mathsf{coll}}(\mathbb{RP}) = |\mathbb{V}|^2/2^L$.

Proof. The proof is deferred to full version. □

The property in Lemma 2 is useful in determining the node index v when one sees an ideal-primitive query related to v. Moreover, it guarantees that each node v maps to a unique ideal-primitive input entry $\mathsf{prelab}(v)$.

<u>Pebbling Legality.</u> Next, we show that with high probability (over the uniform choices of the ideal primitive ip and random coins r), the *ex-post-facto* pebbling $\mathcal{P}(\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)) = (\mathsf{P}_0 = \emptyset, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$ for $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ is *legal*, and thus

$$\sum_{i=0}^{t_{\mathsf{peb}}} |\mathsf{P}_i| \geq \mathsf{cc}(\mathbb{G})$$

as long as the pebbling is *successful.*

Before presenting the lemma, we prove a claim that will be useful in many places.

Claim 1. *Fix any execution* $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ *(with input states* $\sigma_0, \sigma_1, \ldots$*) whose* ex-post-facto pebbling *is* $\mathcal{P}(\mathsf{A}) = (\mathsf{P}_0, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$*. For any* $i \in [t_{\mathsf{peb}}]$ *and any vertex* $v \in \mathsf{P}_i \setminus \mathsf{P}_{i-1}$*, it holds that there is a correct call for* v *in round* i.

Proof. The proof is deferred to full version. □

Lemma 3. *Fix* $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ *(with block length* L*), predecessors-distinct DAG* $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ *(with* n_s *source nodes), non-colliding input vector* $\mathbf{x} \in \{0,1\}^{n_s L}$*, algorithm* A*, and* $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{ip}}\}_{\mathsf{ip} \in \mathbb{IP}}$ *constructed in Sect. 3. With probability at least* $1 - \epsilon_{\mathsf{coll}}(\mathbb{IP}) - \epsilon_{\mathsf{legal}}(\mathbb{IP})$ *(over the uniformly random choices of* ip *and* A*'s internal coins* r*), the pre-labels are distinct and the ex-post-facto pebbling for* $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ *is legal. Here* $\epsilon_{\mathsf{coll}}(\mathbb{IP})$ *is the same as in Lemma 2 and* $\epsilon_{\mathsf{legal}}(\mathbb{CF}) = \mathsf{q} \cdot |\mathbb{V}|/2^{L-1}$*,* $\epsilon_{\mathsf{legal}}(\mathbb{IC}) = \epsilon_{\mathsf{legal}}(\mathbb{RP}) = \mathsf{q} \cdot |\mathbb{V}|/2^{L-2}$*, where* q *is an upper bound on the number of calls made by* A.

Proof. The proof is deferred to full version. □

<u>Pebbling Reduction.</u> Next, we build the connection between the state size and the size of the pebbling configuration. In particular, we show that with high probability, the input state size $|\sigma_i|$ is proportional to $|\mathsf{P}_i|$ for all $i \in \mathbb{N}$.

[16] By non-colliding, we mean $\mathbf{x} = (x_1, \ldots, x_{n_s})$ where $x_i \neq x_j$ for every $i \neq j$.

Lemma 4. *Fix $L = 2^\ell$, predecessors-distinct DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ (with n_s source nodes), non-colliding input vector $\mathbf{x} \in \{0,1\}^{n_s L}$, algorithm A (that makes at most $\mathsf{q}-1$ calls), and $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{ip}}\}_{\mathsf{ip}\in\mathbb{IP}}$ constructed in Sect. 3. Set values $\beta_{\mathbb{CF}} := \lfloor L - 2\log\mathsf{q} - \log|\mathbb{V}| - \log 3\rfloor$, $\beta_{\mathbb{IC}} := \lfloor L - 1 - 2\log\mathsf{q} - \log|\mathbb{V}| - \log 3\rfloor$, and $\beta_{\mathbb{RP}} := \lfloor \frac{L}{2} - 1 - 2\log\mathsf{q} - \log|\mathbb{V}| - \log 3\rfloor$. For any $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ and $\lambda \in \mathbb{N}$, define $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ as the event where the following three conditions* all *hold:*

1. *The pre-labels are distinct from each other.*
2. *The ex-post-facto pebbling $(\mathsf{P}_0, \mathsf{P}_1, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$ for $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ is legal.*
3. *There exists $i \in \mathbb{N}$ such that $|\sigma_i| < |\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda$, where σ_i denotes the input state for round $i+1$ and P_i denotes the pebbling configuration in round i.*

It holds that $\Pr[\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}] \le 2^{-\lambda}$ for all $\lambda \in \mathbb{N}$, where the probability is taken over the choice of $\mathsf{ip} \xleftarrow{\$} \mathbb{IP}$ and random coins of A.

Proof. Without loss of generality we fix r to be the optimal random coins of A that maximizes $\Pr[\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}]$. We will show a predictor P (that hardwires r and has oracle access to ip), such that if $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens (which implies $|\sigma_i| < |\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda$ for some $i \in \mathbb{N}$), there will be a hint h with no more than $|\mathsf{P}_i| \cdot L - \lambda$ (and $|\mathsf{P}_i| \cdot (L-1) - \lambda$ when $\mathbb{IP} = \mathbb{IC}/\mathbb{RP}$) bits, where $P(h)$ can predict $|\mathsf{P}_i|$ ideal primitive entries correctly. Thus by the compression arguments that ideal primitives cannot be compressed (i.e., Lemmas 9 and 10), $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens with probability no more than $2^{-\lambda}$ and the lemma holds. Next we describe the hint h and the predictor P.

<u>The Hint.</u> For any choice of $\mathsf{ip} \in \mathbb{IP}$, if event $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens, there exists a round $i \in \mathbb{N}$ such that $|\sigma_i| < |\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda$, where σ_i is the input state of round $i+1$ and $\mathsf{P}_i = (v_1, v_2, \ldots, v_{|\mathsf{P}_i|})$ is the ex-post-facto pebbling configuration. The hint consists of the state σ_i and the following helper information. (In the following context, if not describe explicitly, by critical call, we always mean a critical call in the trace $\mathsf{A}(\sigma_i)$.)

- A sequence $Q_i = (\mathsf{id}_1, \mathsf{id}_2, \ldots, \mathsf{id}_{|\mathsf{P}_i|}) \in [\mathsf{q}-1]^{|\mathsf{P}_i|}$, where id_j $(1 \le j \le |\mathsf{P}_i|)$ is the index of the *first* critical call for $v_j \in \mathsf{P}_i$ in the trace $\mathsf{A}(\sigma_i)$. (Recall that we sort the calls in chronological order, and assume an implicit order for the calls in the same round.)
- A nodes sequence $W_i = (w_1, w_2, \ldots, w_{|\mathsf{P}_i|})$, where $w_j = v_j$ $(1 \le j \le |\mathsf{P}_i|)$ if the id_j-th call is an output call; otherwise, if the id_j-th call is a correct call for some successor of v_j, then w_j is assigned as the corresponding successor node.
- A sequence $B_i = (b_1, b_2, \ldots, b_{|\mathsf{P}_i|})$, where $b_j \in \{0,1,2\}$ is used to indicate the relation between w_j and v_j. In particular, $w_j = v_j$ if $b_j = 0$, otherwise v_j is the b_j-th predecessor of w_j.
- A sequence $C_i = (\mathsf{cid}_1, \mathsf{cid}_2, \ldots, \mathsf{cid}_{|\mathsf{P}_i|})$, where $\mathsf{cid}_j = 0$ $(1 \le j \le |\mathsf{P}_i|)$ if there is no correct call for $v_j \in \mathsf{P}_i$ in the trace $\mathsf{A}(\sigma_i)$, otherwise cid_j is the query index of the *first* correct call for v_j.

– If $\mathbb{IP} = \mathbb{RP}$, the hint includes an extra sequence $H_i = (h_1, h_2, \ldots, h_{|\mathsf{P}_i|})$, where h_j $(1 \leq j \leq |\mathsf{P}_i|)$ is the label ℓ_{v_j} if there exists some $k > j$ such that $\mathsf{id}_j = \mathsf{id}_k$ (i.e., another node $v_k \in \mathsf{P}_i$ has the same *first* critical call), otherwise h_j is set as empty[17]. We see that there are at most $\lfloor |\mathsf{P}_i|/2 \rfloor$ non-empty values in the sequence, as any ideal-primitive call can be a critical call for at most two vertices.

Note that we can easily recover the configuration P_i from the hint W_i and B_i. The size of the hint is no more than $\mathsf{len}_{\mathbb{IP}} := |\mathsf{P}_i| \cdot L - \lambda$ (and $\mathsf{len}_{\mathbb{IP}} := |\mathsf{P}_i| \cdot (L-1) - \lambda$ when $\mathbb{IP} = \mathbb{IC}/\mathbb{RP}$) bits given the setting of $\beta_{\mathbb{IP}}$.

<u>The Predictor P.</u> Given any input the predictor P parses the input into σ_i, Q_i, W_i, B_i, C_i and H_i as mentioned before[18], and recovers the pebbling configuration P_i. Then P runs the partial execution $\mathsf{A}(\sigma_i)$ and attempts to predict $(\mathsf{prelab}(v), \mathsf{ip}(\mathsf{prelab}(v)))$ for every $v \in \mathsf{P}_i$ *without querying* $\mathsf{ip}(\mathsf{prelab}(v))$. In the following context, if not describe explicitly, by critical call, we always mean a critical call in the trace $\mathsf{A}(\sigma_i)$.

When simulating $\mathsf{A}(\sigma_i)$, the predictor uses the following approach to determine if an ideal-primitive call is a correct call for a node v: The predictor keeps track of the labels $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v for every $v \in \mathbb{V}$. Moreover, after knowing the labels of v's predecessors, the predictor updates $\mathsf{prelab}(v)$ accordingly. Given an ideal-primitive call from A, P determines call correctness by checking the following cases *sequentially*:

– If P knows from hint Q_i that the call is the *first* critical call for some node $v \in \mathsf{P}_i$, then the predictor knows that it is also a correct call for some node w, where w can be extracted from the hint W_i.
– If the call is the first correct call[19] for some node $v \in \mathsf{P}_i$, then P will know it from the hint C_i.
– The call is a *forward* call. Then the predictor checks if there exists a node $v \in \mathbb{V}$ where $\mathsf{prelab}(v)$ was updated and $\mathsf{prelab}(v)$ matches the call input x_c. If so, P asserts that it is a correct call for v.
– $\mathbb{IP} = \mathbb{IC}/\mathbb{RP}$ and the call is an *inverse call.* Then the predictor first checks if there is a node $v \in \mathbb{V}$ where $\mathsf{aftlab}(v)$ was updated and $\mathsf{aftlab}(v)$ matches the call input x_c. If no such v exists, P queries the oracle, and checks if the answer is consistent with some updated $\mathsf{prelab}(v)$ $(v \in \mathbb{V})$.
– If one of the above checks succeed, then after recognizing the correct call for v, P updates $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v accordingly.

Claim 2. *Fix execution* $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ *and round* i*, suppose the pre-labels are distinct, and the ex-post-facto pebbling* $(\mathsf{P}_0, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$ *is legal. For any round* $\gamma > i$*, assume the predictor successfully extracted* ℓ*-labels for all (first) critical calls (in* $\mathsf{A}(\sigma_i)$*) between round* i *and round* γ*. Then for any vertex* $v \in \mathsf{P}_i \cup \cdots \cup \mathsf{P}_\gamma$*, and any*

[17] Note that we don't need an indicator (e.g., $h_j = \perp$) to tell if h_j is empty or not, as we can know it from the sequence Q_i. This enables us to have a shorter H_i.
[18] We assume that the encoding of the hint is unambiguous.
[19] Recall that there is an implicit chronological order for the calls.

correct call for v in round γ (denote the call as $C(v)$), the predictor will correctly recognize the call $C(v)$ when simulating $\mathsf{A}(\sigma_i)$.

Proof. The proof is deferred to full version. □

Next we show how to simulate $\mathsf{A}(\sigma_i)$ and predict ideal-primitive entries.

The predictor simulates $\mathsf{A}(\sigma_i)$ (which corresponds to the partial execution of A after round i) and keeps track of the labels $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v for every $v \in \mathbb{V}$. For each round $\gamma > i$, after receiving the calls from A, the predictor P does follows *sequentially.*

1. *Handling critical calls:* P first enumerates node $v_j \in \mathsf{P}_i$ according to *reverse topological order*[20] and checks the following: If the id_j-th call (i.e. v_j's *first* critical call) is in round γ and ℓ_{v_j} is unknown yet, the predictor uses the hint to extract the label ℓ_{v_j}. The extraction from a critical output call is trivial, thus we assume that the call is an ideal-primitive call. From the hint, the predictor knows that it is a correct ideal primitive call for a node $w_j \in \mathsf{succ}(v_j)$ where w_j can be extracted from the hint W_i. (Note that $w_j \in \mathsf{P}_\gamma$ by definition of critical calls.) The predictor first extracts $\mathsf{prelab}(w_j)$ from the call input/output:
   - If the call is forward, $\mathsf{prelab}(w_j)$ can be identified from the call input.
   - If $w_j \in \mathsf{P}_i$ and the call is an inverse call, since P chooses nodes in P_i according to reverse topological order, and in $\mathsf{A}(\sigma_i)$ the first critical call for w_j appears no later than any correct call for w_j, P must have already extracted ℓ_{w_j}, and thus the predictor can extract $\mathsf{prelab}(w_j)$ from ℓ_{w_j} and the call input without querying the oracle.
   - If $w_j \notin \mathsf{P}_i$ and the call is an inverse call, P can query the oracle and extract the information of $\mathsf{prelab}(w_j)$ from the oracle answer.

   Given w_j and $\mathsf{prelab}(w_j)$, if $\mathbb{IP} = \mathbb{CF}$ or $\mathbb{IP} = \mathbb{IC}$, P can directly extract the label ℓ_{v_j} from $\mathsf{prelab}(w_j)$ and b_j; if $\mathbb{IP} = \mathbb{RP}$ and v_j is the only predecessor of w_j, P can extract $\ell_{v_j} = \mathsf{prelab}(w_j)$; if $\mathbb{IP} = \mathbb{RP}$ and w_j has another predecessor u, we argue that ℓ_u was already known and thus the predictor can obtain the label $\ell_{v_j} = \mathsf{prelab}(w_j) \oplus \ell_u$.
   - If $u \notin \mathsf{P}_i$, since the ex-post-facto pebbling is legal and $w_j \in \mathsf{P}_\gamma$, there exists a round γ' ($i < \gamma' < r$) such that $u \in \mathsf{P}_{\gamma'} \setminus \mathsf{P}_{\gamma'-1}$. By Claim 1, there is a correct call $C(u)$ for u in round γ'. Then by Claim 2, P will recognize the call $C(u)$ and update the label ℓ_u.
   - If u is in P_i but the first critical call for u is before round γ (but after round i), then ℓ_u was already known before round γ.
   - If u equals some node $v_k \in \mathsf{P}_i$ such that v_k and v_j have the same *first* critical call, since ℓ_{v_j} was unknown, it must be the case that $k < j$ and $h_k = \ell_{v_k}$, hence the predictors knew ℓ_u initially from the hint H_i.
2. *Handling correct calls for* P_i: For each node $v_j \in \mathsf{P}_i$ and each correct ideal-primitive call for v_j (note that by Claim 2, P correctly recognizes the call, as the ℓ-labels of (first) critical calls upto round γ were correctly extracted),

[20] $v_{|\mathsf{P}_i|}$ is picked first, then $v_{|\mathsf{P}_i|-1}$,..., and finally v_1.

since the predictor already knew ℓ_{v_j} after handling the first critical call for v_j,[21] she can answer the call *without* querying the ideal primitive:

- If $\mathbb{IP} = \mathbb{CF}$, then ℓ_{v_j} is the query answer.
- If $\mathbb{IP} = \mathbb{IC}$ and the call input has the value (k, x) where $k \in \{0,1\}^L \cup \{\perp\}$ and $x \in \{0,1\}^L$, the answer is $\ell_{v_j} \oplus x$ because $\ell_{v_j} = x \oplus \mathsf{ip}(k, x)$ for a forward call and $\ell_{v_j} = x \oplus \mathsf{ip}^{-1}(k, x)$ for an inverse call.
- If $\mathbb{IP} = \mathbb{RP}$ and the call input has the value x, the answer is $\ell_{v_j} \oplus x$ because $\ell_{v_j} = x \oplus \mathsf{ip}(x)$ for a forward call and $\ell_{v_j} = x \oplus \mathsf{ip}^{-1}(x)$ for an inverse call.

For each round $\gamma > i$, after checking correct/critical calls for all nodes in P_i, the predictor answers the other unanswered calls by making queries to the ideal primitive. Note that in round γ, for every node $v \in \mathsf{P}_i \cup \cdots \cup \mathsf{P}_\gamma$, if there is a correct ideal-primitive call for v, since P already extracted ℓ-labels for all (first) critical calls upto round γ, by Claim 2, P will recognize the call, get the call answer, then update the labels $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$, ℓ_v and the pre-labels of v's successors.

After executing $\mathsf{A}(\sigma_i)$, the predictor will compute $\mathsf{prelab}(v)$ for every $v \in \mathbb{V}$ according to topological order, and predict $\mathsf{ip}(\mathsf{prelab}(v))$ for every $v \in \mathsf{P}_i$. In particular, if $\mathbb{IP} = \mathbb{CF}$, $\mathsf{ip}(\mathsf{prelab}(v)) = \ell_v$; if $\mathbb{IP} = \mathbb{IC}$, let x be the last L-bit string of $\mathsf{prelab}(v)$, then $\mathsf{ip}(\mathsf{prelab}(v)) = x \oplus \ell_v$; if $\mathbb{IP} = \mathbb{RP}$, then $\mathsf{ip}(\mathsf{prelab}(v)) = \mathsf{prelab}(v) \oplus \ell_v$. Note that if $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens and the input is the hint h mentioned above, the predictor does not need to query $\mathsf{ip}(\mathsf{prelab}(v))$ for any $v \in \mathsf{P}_i$ as the answer can be computed from $\mathsf{prelab}(v)$ and the extracted label ℓ_v.

<u>Correctness of the Predictor.</u> If $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens and P's input is the hint mentioned above, the predictor will correctly predict $(\mathsf{prelab}(v), \mathsf{ip}(\mathsf{prelab}(v)))$ for every $v \in \mathsf{P}_i$ without querying $\mathsf{ip}(\mathsf{prelab}(v))$. Recall that $\{\mathsf{prelab}(v)\}$ are distinct so that P also predicts $|\mathsf{P}_i|$ ideal-primitive entries.

First, we note that the labels being updated (including $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v for $v \in \mathbb{V}$) are correct by induction on the time order of updating. Initially, only the pre-labels of source vertices were updated which are correct. Assume all the labels being updated are correct up to now. A new label $\mathsf{prelab}(v)$ (or $\mathsf{aftlab}(v)$) will be updated because one of the following possibilities:

1. P recognizes the *first* correct call for $v \in \mathsf{P}_i$ from the hint C_i (and thus correct by the hint).
2. P recognizes a correct call for v by finding out that the call input/output matches the previously updated $\mathsf{aftlab}(v)$ (or $\mathsf{prelab}(v)$) which is correct by inductive hypothesis.
3. P computes the label according to topological order (at the end).
4. $\mathsf{prelab}(v)$ is updated because the ℓ-labels of v's predecessors were all updated previously (which are correct by inductive hypothesis).

Similarly, a new label ℓ_v will be updated either because the possibility 2 as above, or because P extracts ℓ_v from the *first* critical call for v (and thus correct by

[21] Recall that in $\mathsf{A}(\sigma_i)$, there was no correct call for v before the round of the first critical call for v.

the hint). Note that the argument above also implies that P will not output an incorrect prediction.

It remains to prove that P will never query $\mathsf{ip}(\mathsf{prelab}(v))$ for any $v \in \mathsf{P}_i$. First, when simulating $\mathsf{A}(\sigma_{\sigma_i})$, P will recognize the first correct call of v from the hint C_i and answer the call using the extracted label ℓ_v. Then $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v will all be updated. For the following correct calls, since $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v have been updated, P will recognize and answer the call without querying ip. Lastly, when computing $\mathsf{prelab}(v)$ for $v \in \mathbb{V}$ according to topological order, P will not query $\mathsf{ip}(\mathsf{prelab}(v))$ for any $v \in \mathsf{P}_i$ as the answer will be computed from $\mathsf{prelab}(v)$ and the extracted label ℓ_v.

In summary, with probability at least $\Pr[\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}]$, there exists a short hint h where $P(h)$ correctly guesses $|\mathsf{P}_i|$ ideal-primitive entries, thus by Lemmas 9 and 10, we have $\Pr[\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}] \leq 2^{-\lambda}$ and the lemma holds. □

Putting All Things Together. For an execution $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$, we say $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$ is correct if the algorithm generates the correct graph function output at the end; we say $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$ is lucky if it is correct but there is a vertex $v \in \mathsf{sink}$ where A did not make any correct call for v before outputting the label ℓ_v. Note that if $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$ is correct but not lucky, the ex-post-facto pebbling will be successful. Moreover, with similar compression argument as in Lemmas 2 and 3, the probability (over the uniform choice of ip and A's internal coins) that A is lucky is no more than $\epsilon_{\mathsf{luck}}(\mathbb{IP})$, where $\epsilon_{\mathsf{luck}}(\mathbb{CF}) = |\mathbb{V}|/2^L$ and $\epsilon_{\mathsf{luck}}(\mathbb{IC}) = \epsilon_{\mathsf{luck}}(\mathbb{RP}) = |\mathbb{V}|/2^{L-1}$.

In summary, for any algorithm A that correctly computes the graph function with probability $\epsilon_{\mathsf{A}} > 2 \cdot (\epsilon_{\mathsf{coll}}(\mathbb{IP}) + \epsilon_{\mathsf{legal}}(\mathbb{IP}) + \epsilon_{\mathsf{luck}}(\mathbb{IP}))$, we set $\lambda \in \mathbb{N}$ as the minimal integer such that $\epsilon(\lambda) = \epsilon_{\mathsf{coll}}(\mathbb{IP}) + \epsilon_{\mathsf{legal}}(\mathbb{IP}) + \epsilon_{\mathsf{luck}}(\mathbb{IP}) + 2^{-\lambda} \leq \epsilon_{\mathsf{A}}/2$. Then the following conditions hold with probability more than $\epsilon_{\mathsf{A}} - \epsilon(\lambda) \geq \epsilon_{\mathsf{A}}/2$:

1. The pre-labels are distinct from each other.
2. The ex-post-facto pebbling is legal and successful, hence
$$\sum_{i=1}^{t_{\mathsf{peb}}} |\mathsf{P}_i| \geq \mathsf{cc}(\mathbb{G}).$$
3. For every $i \in [t_{\mathsf{peb}}]$, it holds that $|\sigma_i| \geq |\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda$. Here $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ terminates at round $t_{\mathsf{peb}} + 1$, σ_i is the input state for round $i + 1$, and P_i is the pebbling configuration in round i.
Thus we have
$$\begin{aligned}\mathsf{CMC}(\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)) &\geq \sum_{i=1}^{t_{\mathsf{peb}}} |\sigma_i| \geq \sum_{i=1}^{t_{\mathsf{peb}}} (|\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda) \\ &\geq \left(\sum_{i=1}^{t_{\mathsf{peb}}} |\mathsf{P}_i|\right) \cdot \beta_{\mathbb{IP}} - t_{\mathsf{peb}} \cdot \lambda \\ &\geq \mathsf{cc}(\mathbb{G}) \cdot \beta_{\mathbb{IP}} - t_{\mathsf{peb}} \cdot \lambda \geq \mathsf{cc}(\mathbb{G}) \cdot \beta_{\mathbb{IP},\lambda},\end{aligned}$$
where $\beta_{\mathbb{IP},\lambda} = \beta_{\mathbb{IP}} - \lambda$ as $\mathsf{cc}(\mathbb{G}) \geq t_{\mathsf{peb}}$.

Therefore we have

$$\mathbb{E}[\mathsf{CMC}(\mathsf{A}^{\mathsf{ip}}(\mathbf{x};r))] \geq \frac{\epsilon_{\mathsf{A}}}{2} \cdot \beta_{\mathbb{IP},\lambda} \cdot \mathsf{cc}(\mathbb{G}).$$

By plugging in the corresponding $\epsilon_{\mathsf{coll}}(\mathbb{IP})$, $\epsilon_{\mathsf{legal}}(\mathbb{IP})$, $\epsilon_{\mathsf{luck}}(\mathbb{IP})$ and $\beta_{\mathbb{IP}}$ for the ideal primitive $\mathbb{IP}$, we can find the optimal parameter $\lambda_{\mathbb{IP}}$, and compute

$$\beta(\epsilon_{\mathsf{A}}, \log|\mathbb{V}|) = \frac{\epsilon_{\mathsf{A}}}{2} \cdot (\beta_{\mathbb{IP}} - \lambda_{\mathbb{IP}}),$$

which leads to Theorems 1, 2 and 3.

□

3.3 iMHFs from Small-Block Labeling Functions

In this section, from any graph, we construct graph-based iMHFs from the small-block labeling functions built in Sect. 3.1.

Proposition 1. *Fix $L = 2^{\ell}$ and let $\mathcal{H}_{\mathsf{fix}}$ be the β-small-block labeling function built in Sect. 3.1. For any 2-indegree (predecessors-distinct) DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with $N = 2^n$ vertices and single source/sink, the graph labeling functions $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ is $(\mathsf{C}^{\|}_{\mathcal{F}}, \Delta_{\mathcal{F}}, N)$-memory hard, where for all $\epsilon \in [3 \cdot 2^{-L/10}, 1]$, it holds that*

$$\mathsf{C}^{\|}_{\mathcal{F}}(\epsilon) \geq \Omega(\epsilon \cdot \mathsf{cc}(\mathbb{G}) \cdot L), \qquad \Delta_{\mathcal{F}}(\epsilon) \leq O\left(\frac{\mathsf{st}(\mathbb{G}, N)}{\mathsf{cc}(\mathbb{G})}\right).$$

Proof. The proof is deferred to full version. □

The graph $\mathbb{G}$ in [3] has pebbling complexities $\mathsf{cc}(\mathbb{G}) = \Omega(N^2/\log N)$ and $\mathsf{st}(\mathbb{G}, N) = O(N^2/\log N)$, thus we obtain the following corollary.

Corollary 1. *Fix $L = 2^{\ell}$ and let $\mathcal{H}_{\mathsf{fix}}$ be the β-small-block labeling function built in Sect. 3.1. Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be the 2-indegree (predecessors-distinct) DAG in [3] (with $N = 2^n$ vertices). The graph labeling functions $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ is $(\mathsf{C}^{\|}_{\mathcal{F}}, \Delta_{\mathcal{F}}, N)$-memory hard, where for all $\epsilon \in [3 \cdot 2^{-L/10}, 1]$, it holds that*

$$\mathsf{C}^{\|}_{\mathcal{F}}(\epsilon) \geq \Omega\left(\frac{\epsilon \cdot N^2 \cdot L}{\log N}\right), \qquad \Delta_{\mathcal{F}}(\epsilon) \leq O(1).$$

4 MHFs from Wide-Block Labeling Functions

Ideally, we target a CMC which is as high as possible, while keeping the evaluation of the function within a feasible margin for the legitimate users. An option is to use a bigger graph with high CC and small-block labeling functions. However, this can lead to large description size. A way out here is to choose a graph family that has succinct description. Unfortunately, as far as we know, practical hard-to-pebble graphs are randomly sampled and do not have a succinct description of the actual graph, only of the sampling process. To reduce description complexity of MHFs, in this section, we construct a family of graph-based iMHFs based on an abstraction called *wide-block labeling functions*. In Sect. 4.1, we define and construct a family of wide-block labeling functions from small-block labeling functions. In Sect. 4.2, we prove that the construction satisfies pebbling reducibility with respect to depth-robust graphs. Finally in Sect. 4.3, we construct succinct iMHFs from wide-block labeling functions.

4.1 Wide-Block Labeling Functions: Definition and Construction

Definition 7 (Wide-Block Labeling Functions). *For any ideal primitive* $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$*,* $\delta \in \mathbb{N}$ *and* $W = 2^w$*, we say*

$$\mathcal{H}_{\delta,w} = \left\{ \mathsf{vlab}^{\mathsf{ip}}_{\gamma,w} : \{0,1\}^{\gamma W} \rightarrow \{0,1\}^{W} \right\}_{\mathsf{ip} \in \mathbb{IP}, 1 \leq \gamma \leq \delta}$$

is a family of $\beta_{\delta,w}$*-wide-block labeling functions if it satisfies the following property.*

$\beta_{\delta,w}$**-pebbling reducibility w.r.t. depth-robust graphs:** *For any* $\epsilon \in [0,1]$ *and any* δ*-indegree* (e,d)*-depth robust (first-predecessor-distinct) DAG* $\mathbb{G} = (\mathbb{V}, \mathbb{E})$[22]*, the graph functions* $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ *satisfies*

$$\mathsf{CMC}_{\epsilon}(\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}) \geq e \cdot (d-1) \cdot \beta_{\delta,w}(\epsilon, \log |\mathbb{V}|),$$

where $\mathsf{CMC}_{\epsilon}(\cdot)$ *is* ϵ*-cumulative-memory-complexity (Definition 1).*

<u>CONSTRUCTION.</u> Next we show how to construct wide-block labeling functions from small-block labeling functions. The construction is the composition of two graph functions MIX and SSDR, which can be built from any small-block labeling functions.

Remark 5. There are tailored-made variable-length hash functions available in the real-world, for example within Scrypt [19,20] and Argon2 [10]. However, even by modeling the underlying block/stream cipher as an ideal primitive, we do not know how to prove the pebbling-reducibility of the hash functions. Hence we seek another construction of labeling functions.

In the following context, we fix the indegree parameter $\delta \in \mathbb{N}$, ideal primitive length $L = 2^{\ell}$, output length $W = 2^w$ of the wide-block labeling functions, and denote as $K = 2^k := W/L$ the ratio between W and L. We will omit these variables in notation when it is clear in the context.

We show how to construct the family of labeling functions $\mathcal{H}_{\delta,w}$. For any $1 \leq \gamma \leq \delta$, and any ideal primitive $\mathsf{ip} \in \mathbb{IP}$, we define the labeling function $\mathsf{vlab}^{\mathsf{ip}}_{\gamma,w} : \{0,1\}^{\gamma W} \rightarrow \{0,1\}^{W}$ as the composition of two functions, namely,

[22] $\mathbb{G}$ has a single source/sink vertex.

$\mathsf{mix}^{\mathsf{ip}}_{\gamma} : \{0,1\}^{\gamma W} \rightarrow \{0,1\}^{W}$ and $\mathsf{ssdr}^{\mathsf{ip}}_{\delta} : \{0,1\}^{W} \rightarrow \{0,1\}^{W}$. More precisely, for an input vector $\mathbf{x} \in \{0,1\}^{\gamma W}$, we define the W-bit function output as

$$\mathsf{vlab}^{\mathsf{ip}}_{\gamma,w}(\mathbf{x}) := \mathsf{ssdr}^{\mathsf{ip}}_{\delta}(\mathsf{mix}^{\mathsf{ip}}_{\gamma}(\mathbf{x})).$$

Next, we specify the functions $\mathsf{mix}^{\mathsf{ip}}_{\gamma}$ and $\mathsf{ssdr}^{\mathsf{ip}}_{\delta}$.

<u>COMPONENT: MIX FUNCTIONS.</u> Denote as $\mathsf{flab}^{\mathsf{ip}} : \{0,1\}^{L} \cup \{0,1\}^{2L} \rightarrow \{0,1\}^{L}$ a small-block labeling function (Definition 6), and let $K := W/L$ be the ratio between W and L. We define

$$\mathsf{mix}^{\mathsf{ip}}_{\gamma} := \mathsf{F}^{\mathsf{ip}}_{\mathbb{G}^{\gamma,K}_{\mathsf{mix}}} : \{0,1\}^{\gamma W} \rightarrow \{0,1\}^{W=KL}$$

as the graph function (Sect. 2.3) built upon a DAG $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}$ and the labeling function $\mathsf{flab}^{\mathsf{ip}}$. (Note that we can use $\mathsf{flab}^{\mathsf{ip}}$ as the labeling function since the maximal indegree of $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}$ is 2.) The graph $\mathbb{G}^{\gamma,K}_{\mathsf{mix}} = (\mathbb{V}^{\gamma,K}_{\mathsf{mix}}, \mathbb{E}^{\gamma,K}_{\mathsf{mix}})$ is defined as follows.

Nodes set: The set $\mathbb{V}^{\gamma,K}_{\mathsf{mix}}$ has γK source nodes (which represent the γK input blocks), and we use $\langle 0,j\rangle$ $(1 \leq j \leq \gamma K)$ to denote the jth source node. Besides, there are γK columns each with K nodes. We use $\langle i,j\rangle$ $(1 \leq i \leq \gamma K,$ $1 \leq j \leq K)$ to denote the node at the ith column and jth row. The K nodes at the last column are the sink nodes (which represent the K output blocks).

Edges set: The set $\mathbb{E}^{\gamma,K}_{\mathsf{mix}}$ has $\gamma K^2 + \gamma(K-1)K + K - 1$ edges. Each source node $\langle 0,i\rangle$ $(1 \leq i \leq \gamma K)$ has K outgoing edges to the K nodes of column i, namely, $\{\langle i,j\rangle\}_{j\in[K]}$. For each column i $(1 \leq i < \gamma K)$ and each row j $(1 \leq j \leq K)$, the node $\langle i,j\rangle$ has an outgoing edge to $\langle i+1,j\rangle$ at the next column. Finally, each source node $\langle 0,j\rangle$ $(2 \leq j \leq K)$ has an outgoing edge to $\langle 1,j\rangle$[23] (Fig. 1).

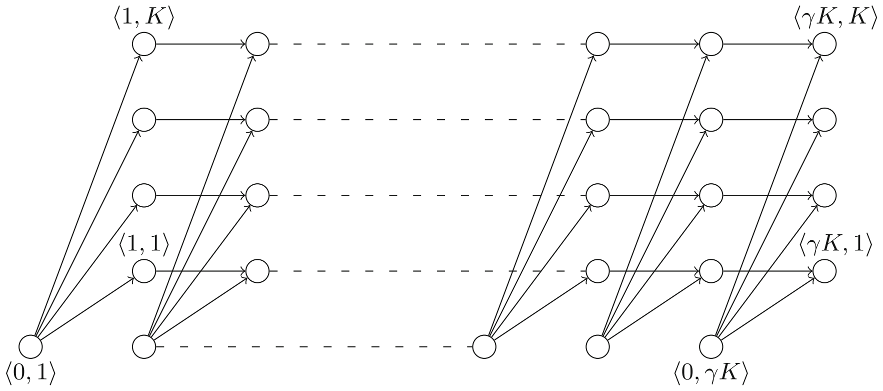
$\langle 1,K\rangle$
$\langle \gamma K,K\rangle$
$\langle 1,1\rangle$
$\langle \gamma K,1\rangle$
$\langle 0,1\rangle$
$\langle 0,\gamma K\rangle$

Fig. 1. The graph $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}$ for $K = 4$. We omitted the edges from $\langle 0,j\rangle$ to $\langle 1,j\rangle$ $(2 \leq j \leq K)$ for clarity of the figure.

[23] The last $K-1$ edges make sure that the K nodes at column 1 have distinct sets of predecessors (Sect. 2.3).

COMPONENT: SSDR FUNCTIONS. Denote as $\mathsf{flab}^{\mathsf{ip}} : \{0,1\}^L \cup \{0,1\}^{2L} \to \{0,1\}^L$ a small-block labeling function (Definition 6), and let $K := W/L$ be the ratio between W and L. Fix $\delta \in \mathbb{N}$, we define

$$\mathsf{ssdr}_\delta^{\mathsf{ip}} := \mathsf{F}^{\mathsf{ip}}_{\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}} : \{0,1\}^W \to \{0,1\}^W$$

as the graph function built upon the labeling function $\mathsf{flab}^{\mathsf{ip}}$ and a DAG $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$. (Note that we can use $\mathsf{flab}^{\mathsf{ip}}$ as the labeling function since the maximal indegree of $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$ is 2.) $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}} = (\mathbb{V}^{\delta,K}_{\mathsf{ssdr}}, \mathbb{E}^{\delta,K}_{\mathsf{ssdr}})$ is a *source-to-sink-depth-robust* graph (Definition 4) defined as follows.

Nodes set: The set $\mathbb{V}^{\delta,K}_{\mathsf{ssdr}}$ has $K(1+\delta K)$ vertices distributing across $1+\delta K$ columns and K rows. For every $i \in \{0,\ldots,\delta K\}$ and every $j \in \{1,\ldots,K\}$, we use $\langle i,j \rangle$ to denote the node at column i and row j. The K nodes at column 0 are the source nodes and the K nodes at column δK are the sink nodes.

Edges set: The set $\mathbb{E}^{\delta,K}_{\mathsf{ssdr}}$ consists of 3 types of edges. The first type is called *horizontal edges*: For every i $(0 \le i < \delta K)$ and every j $(1 \le j \le K)$, there is an edge from node $\langle i,j \rangle$ to node $\langle i+1,j \rangle$. The second type is called *vertical edges*: For every i $(2 \le i \le \delta K)$ and every j $(1 \le j < K)$, there is an edge from node $\langle i,j \rangle$ to node $\langle i,j+1 \rangle$. The third type is called *backward edges*: For every j $(1 \le j < K)$, there is an edge from node $\langle \delta K,j \rangle$ to node $\langle 1,j+1 \rangle$. In total, there are $\delta K^2 + \delta K \cdot (K-1) < 2\delta K^2$ edges (Fig. 2).

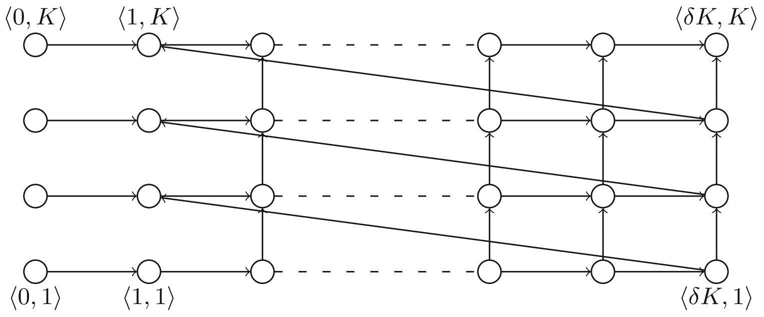

Fig. 2. The graph $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$ for $K = 4$

We prove a useful lemma showing that $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$ is *source-to-sink-depth-robust.*

Lemma 5. *Fix any $K = 2^k \ge 4$ and $\delta \in \mathbb{N}$, the graph $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$ is $(\frac{K}{4}, \frac{\delta K^2}{2})$-source-to-sink-depth-robust.*

Proof. The proof is deferred to full version. □

4.2 Wide-Block Labeling Functions: Pebbling Reducibility

In this section, we show that the labeling functions constructed in Sect. 4.1 satisfy pebbling reducibility with respect to (first-predecessor-distinct) depth-robust graphs. We will make use of the following notation.

<u>Graph Composition:</u> Given a graph $\mathbb{G}_1$ (with n_1 source nodes and n_2 sink nodes), and a graph $\mathbb{G}_2$ (with n_2 source nodes and n_3 sink nodes), we define $\mathbb{G}_1 \circ \mathbb{G}_2$ as the composition of $\mathbb{G}_1$ and $\mathbb{G}_2$, namely, we merge the ith sink node of $\mathbb{G}_1$ with the ith source node of $\mathbb{G}_2$ for each $i \in [n_2]$[24], and take the union of the rest parts of the graphs.

Theorem 4. *Fix $L = 2^\ell$, $W = 2^w \geq L$, and set $K := W/L$. Let $\mathcal{H}_{\mathsf{fix}}$ be any β_{fix}-small-block labeling functions. For any $\delta \in \mathbb{N}$, the labeling functions $\mathcal{H}_{\delta,w}$ constructed in Sect. 4.1 is $\beta_{\delta,w}$-pebbling-reducible w.r.t. (first-predecessor-distinct[25]) depth-robust graphs (with single source/sink) where*

$$\beta_{\delta,w}(\epsilon, \log|\mathbb{V}|) \geq \frac{\delta K^3}{8} \cdot \beta_{\mathsf{fix}}(\epsilon, \log|\mathbb{V}|).$$

Here $\epsilon \in (0,1]$ and $|\mathbb{V}|$ is the number of vertices in the graph.

Remark 6 (Generalization). For the wide-block labeling functions constructed in Sect. 4.1, we make use of a specific graph $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K}$ that is source-to-sink depth robust. We emphasize, however, that any source-to-sink depth robust graphs suffice. In particular, by replacing $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K}$ with any 2-indegree DAG $\mathbb{G}^*$ where i) $\mathbb{G}^*$ has K source/sink nodes and ii) $\mathbb{G}^*$ is (e^*, d^*)-source-to-sink depth robust, the corresponding wide-block labeling functions is still β-pebbling-reducible, where

$$\beta(\epsilon, \log|\mathbb{V}|) \geq e^* \cdot d^* \cdot \beta_{\mathsf{fix}}(\epsilon, \log|\mathbb{V}|).$$

We leave finding new source-to-sink depth-robust graphs as an interesting direction for future work.

Remark 7. Note that in Theorem 4, the pebbling reducibility only holds for depth-robust graphs. It is hard to directly link CMC and $\mathsf{cc}(\mathbb{G})$. More discussions can be found at Remark 2.

Proof (of Theorem 4). Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be any (first-predecessor-distinct) (e,d)-depth-robust DAG with δ-indegree and single source/sink, let $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ be the graph functions built upon $\mathbb{G}$ and $\mathcal{H}_{\delta,w}$. It is sufficient to show that for every $\epsilon \in (0,1]$,

$$\mathsf{CMC}_\epsilon(\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}) \geq \beta_{\mathsf{fix}}(\epsilon, \log|\mathbb{V}|) \cdot \frac{\delta K^3}{8} \cdot e \cdot (d-1).$$

By opening the underlying graph structure of $\mathcal{H}_{\delta,w}$, we see that $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ is also a graph function built upon functions $\mathcal{H}_{\mathsf{fix}}$ and an extension graph $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ that has the following properties.

- **Nodes Expansion:** Every node $v \in \mathbb{V}$ in the original graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ is expanded into K nodes, that is,

$$\mathsf{copy}(v) := \left(v^{(1)}, \ldots, v^{(K)}\right).$$

[24] We assume an implicit order for nodes in $\mathbb{G}_1$ and $\mathbb{G}_2$.
[25] See Remark 3 for definition of first-predecessor-distinctness.

– **Neighborhood Connection:** For every *non-source* node $v \in \mathbb{V} - \mathsf{src}(\mathbb{G})$, denote as $\mathsf{pred}(v) := (u_1, \ldots, u_\gamma)$ the predecessors of v in $\mathbb{G}$. In $\mathsf{Ext}_{\delta,K}(\mathbb{G})$, there is a subgraph $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v)$ that connects

$$\mathsf{neighbor}(v) := \{\mathsf{copy}(u_1), \ldots, \mathsf{copy}(u_\gamma)\}$$

to the set $\mathsf{copy}(v)$, where $\mathsf{neighbor}(v)$ (and $\mathsf{copy}(v)$) are the source nodes (and the sink nodes) of $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v)$, respectively. Note $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v)$ has the identical graph structure with the composition of $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}$ and $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$.

By first-predecessor-distinctness of $\mathbb{G}$ and by the graph structure of the MIX graph, it holds that $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ is a predecessors-distinct graph with 2-indegree. Next, we will show that the extension graph $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ is $(e, (d-1)\cdot\delta K)$-depth-robust for $K \in \{1, 2\}$ and $(eK/4, (d-1)\cdot\delta K^2/2)$-depth-robust for $K \geq 4$. By Lemma 1, for any $K = 2^k \geq 1$, we have

$$\mathsf{cc}(\mathsf{Ext}_{\delta,K}(\mathbb{G})) \geq \frac{\delta K^3}{8} \cdot e \cdot (d-1).$$

Thus by β_{fix}-pebbling reducibility of $\mathcal{H}_{\mathsf{fix}}$[26], we obtain Theorem 4.

Before proving the depth-robustness of the extension graph, we introduce a useful notation called *meta-node* [3]. Intuitively, meta-node maps each vertex of the original graph $\mathbb{G}$ to a set of vertices in $\mathsf{Ext}_{\delta,K}(\mathbb{G})$.

<u>Meta-Node:</u> We define meta-node $\mathsf{nodes}(v)$ for every node $v \in \mathbb{V}$: For every *non-source* node $v \in \mathbb{V} - \mathsf{src}(\mathbb{G})$, we define $\mathsf{nodes}(v)$ as the set of vertices in the graph $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v) - \mathsf{neighbor}(v)$; for every *source* node $v \in \mathsf{src}(\mathbb{G})$, we define $\mathsf{nodes}(v) := \mathsf{copy}(v)$. Note that for any $u, v \in \mathbb{V}$ such that $u \neq v$, the sets $\mathsf{nodes}(u)$ and $\mathsf{nodes}(v)$ are disjoint.

<u>Depth-robustness of the Extension Graph:</u> Next we show the depth robustness of the extension graph. We first consider the simpler case where $K \in \{1, 2\}$. (In the following context, for a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, we sometimes think $\mathbb{G} = \mathbb{V} \cup \mathbb{E}$ as the union of set $\mathbb{V}$ and $\mathbb{E}$ if there is no ambiguity.)

Lemma 6. *For any $K = 2^k \in \{1, 2\}$ and (e, d)-depth robust DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ that has maximal indegree $\delta \in \mathbb{N}$, the corresponding extension graph $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ is $(e, (d-1)\cdot\delta K)$-depth-robust.*

Proof. The proof is deferred to full version. □

Next, we consider the more general case where $K \geq 4$.

Lemma 7. *For any $K = 2^k \geq 4$ and any (e, d)-depth robust DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with maximal indegree $\delta \in \mathbb{N}$, the corresponding extension graph $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ is $(\frac{K}{4} \cdot e, \frac{\delta K^2}{2} \cdot (d-1))$-depth-robust.*

[26] Note that β_{fix}-pebbling reducibility holds for multi-sources graphs.

Proof. For any nodes subset $S_{\mathsf{ext}} \subseteq \mathsf{Ext}_{\delta,K}(\mathbb{G})$ such that $|S_{\mathsf{ext}}| \leq \frac{K}{4} \cdot e$, we show that $\mathsf{depth}(\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}) \geq \frac{\delta K^2}{2} \cdot (d-1)$, which finishes the proof.

<u>STEP 1:</u> From S_{ext}, we first derive a set of nodes $S \subseteq \mathbb{V}$ in the graph $\mathbb{G}$, and find a long path in $\mathbb{G} - S$.

Claim 3. *Define a set*

$$S := \left\{ v \in \mathbb{V} : |\mathsf{nodes}(v) \cap S_{\mathsf{ext}}| \geq \frac{K}{4} \right\},$$

there exists a d-path[27] $P = (v_1, \ldots, v_d)$ *in the graph* $\mathbb{G} - S$.

Proof. The proof is deferred to full version. □

<u>STEP 2:</u> Given the path $P = (v_1, \ldots, v_d) \subseteq \mathbb{G} - S$, next in Lemma 8 we show that for every $i \in [d-1]$, there exists a long path from $\mathsf{copy}(v_i)$ to $\mathsf{copy}(v_{i+1})$ in the graph $\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}$, then by connecting the $d-1$ paths, we obtain a path with length at least $\frac{\delta K^2}{2} \cdot (d-1)$, hence finish the proof of Lemma 7. Note that the path extraction from $\mathsf{copy}(v_i)$ to $\mathsf{copy}(v_{i+1})$ consists of two steps: First we exploit the structure of SSDR graphs and obtain a long path ending at a node in $\mathsf{copy}(v_{i+1})$, then we exploit the structure of MIX graphs and connect $\mathsf{copy}(v_i)$ to the source node of the obtained path.

Lemma 8. *Given the path* $P = (v_1, \ldots, v_d) \subseteq \mathbb{G} - S$ *(obtained in Claim 3) and the graph* $\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}$*, there exists a nodes sequence* $(u_1, \ldots, u_d)$ *(where* $u_i \in \mathsf{copy}(v_i) - S_{\mathsf{ext}}$ *for every* $i \in [d]$*), such that for every* $i \in [d-1]$*, there is a path (with length at least* $\frac{\delta K^2}{2}$*) that connects* u_i *and* u_{i+1} *in* $\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}$.

Proof. We first show that there exists a path (with length at least $\frac{\delta K^2}{2}$) from some node $u_1 \in \mathsf{copy}(v_1) - S_{\mathsf{ext}}$ to some node $u_2 \in \mathsf{copy}(v_2) - S_{\mathsf{ext}}$. (The arguments for $u_2, \ldots, u_d$ will be similar.) The idea consists of two steps: First, we find a *long* source-to-sink path in $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2)$; second, we connect u_1 to the starting node of the source-to-sink path. We stress that the path we obtain has no intersection with S_{ext}.

<u>FINDING THE SOURCE-TO-SINK PATH:</u> We first define a set S_{ssdr} and find a source-to-sink path in $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - S_{\mathsf{ssdr}}$.

Claim 4. *Define* $\mathsf{row} \subseteq [K]$ *as the set of row indices where* j *is in* row *if and only if the* j*th row of* $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) - \mathsf{neighbor}(v_2)$ *has intersection with* S_{ext}*. Define a set*

$$S_{\mathsf{ssdr}} := \{\langle 0, j\rangle_{\mathsf{ssdr}}\}_{j \in \mathsf{row}} \cup (S_{\mathsf{ext}} \cap \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2)),$$

where $\langle 0, j\rangle_{\mathsf{ssdr}}$ *is the source node of* $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2)$ *at row* j*. The graph* $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - S_{\mathsf{ssdr}}$ *has a source-to-sink path of* $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2)$ *with length at least* $\frac{\delta K^2}{2}$.

[27] A d-path is a path with d vertices.

Proof. The proof is deferred to full version. □

PATHS CONNECTION: Next, we show how to connect $\mathsf{copy}(v_1) - S_{\mathsf{ext}}$ to the starting node of the source-to-sink path. Here we exploit the structure of MIX graphs.

Claim 5. *Denote as u_1 an arbitrary node in the non-empty set*[28] $\mathsf{copy}(v_1) - S_{\mathsf{ext}}$ *and $P_{\mathsf{ssdr}}(v_2)$ the source-to-sink path obtained in Claim 4. The graph $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) - S_{\mathsf{ext}}$ has a path from u_1 to the starting node of $P_{\mathsf{ssdr}}(v_2)$.*

Proof. The proof is deferred to full version. □

Finally, using identical arguments, we can show that for every $i \in \{2, \ldots, d-1\}$, there exists a path (with length at least $\frac{\delta K^2}{2}$) from the node $u_i \in \mathsf{copy}(v_i) - S_{\mathsf{ext}}$ to some node $u_{i+1} \in \mathsf{copy}(v_{i+1}) - S_{\mathsf{ext}}$. Hence we finish the proof of Lemma 8. □

From Lemma 8, we obtain Lemma 7. □

From Lemmas 6 and 7, we obtain Theorem 4. □

4.3 iMHFs from Wide-Block Labeling Functions

In this section, from a relatively *small* depth-robust graph, we construct a graph-based iMHF with *strong* memory hardness based on the wide-block labeling functions built in Sect. 4.1.

Theorem 5. *Fix $L = 2^\ell$, $W = 2^w$ and set $K := W/L$. Let $\mathcal{H}_{\delta,w}$ be the wide-block labeling functions built in Sect. 4.1. For any (first-predecessor-distinct*[29]*) (e,d)-depth-robust DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$*[30] *with δ-indegree and $N = 2^n$ vertices, the graph-based iMHFs family $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ is $(\mathsf{C}^{\|}_{\mathcal{F}}, \Delta_{\mathcal{F}}, 2\delta N K^2)$-memory hard, where for sufficiently large $\epsilon \leq 1$, it holds that*

$$\mathsf{C}^{\|}_{\mathcal{F}}(\epsilon) \geq \Omega(\epsilon \cdot e \cdot d \cdot \delta \cdot K^2 \cdot W), \qquad \Delta_{\mathcal{F}}(\epsilon) \leq O\left(\frac{\mathsf{st}(\mathbb{G}, N)}{e \cdot d}\right).$$

The graph $\mathbb{G}$ in [3] is $(\Omega(N/\log N), \Omega(N))$-depth-robust and satisfies $\mathsf{st}(\mathbb{G}, N) = O(N^2/\log N)$, thus we obtain the following corollary.

Corollary 2. *Fix $L = 2^\ell$, $W = 2^w$ and set $K := W/L$. Let $\mathcal{H}_{2,w}$ be the labeling functions built in Sect. 4.1 and $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be the 2-indegree DAG in [3] (with $N = 2^n$ vertices). The graph labeling functions $\mathcal{F}_{\mathbb{G},\mathcal{H}_{2,w}}$ is $(\mathsf{C}^{\|}_{\mathcal{F}}, \Delta_{\mathcal{F}}, O(NK^2))$-memory hard, where for sufficiently large $\epsilon \leq 1$, it holds that*

$$\mathsf{C}^{\|}_{\mathcal{F}}(\epsilon) \geq \Omega\left(\frac{\epsilon \cdot N^2 \cdot K^2 \cdot W}{\log N}\right), \qquad \Delta_{\mathcal{F}}(\epsilon) \leq O(1).$$

Proof (of Theorem 5). The proof is deferred to full version. □

[28] Note that $\mathsf{copy}(v_1) - S_{\mathsf{ext}}$ is non-empty because $|\mathsf{copy}(v_1) \cap S_{\mathsf{ext}}| \leq |\mathsf{nodes}(v_1) \cap S_{\mathsf{ext}}| < \frac{K}{4} < K = |\mathsf{copy}(v_1)|$. The first inequality holds as $\mathsf{copy}(v_1) \subseteq \mathsf{nodes}(v_1)$, the second inequality holds because $v_1 \notin S$.

[29] See Remark 3 for definition of first-predecessor-distinctness.

[30] $\mathbb{G}$ has single source/sink.

5 Instantiations and Open Problems

We defer instantiations and discussions of future work in full version.

Acknowledgments. The authors were partially supported by NSF grants CNS-1553758 (CAREER), CNS-1719146, CNS-1528178, and IIS-1528041, and by a Sloan Research Fellowship.

A Compression Arguments

Lemma 9 **([16]).** *Fix an algorithm* A*, let* $\mathcal{B}$ *be a sequence of random bits.* A *on input a hint* $h \in \mathcal{H}$ *adaptively queries specific bits of* $\mathcal{B}$ *and outputs* p *indices of* $\mathcal{B}$ *that were not queried before, along with guesses for each of the bits. The probability (over the choice of* $\mathcal{B}$ *and randomness of* A*) that there exists an* $h \in \mathcal{H}$ *where* $\mathsf{A}(h)$ *guesses all bits correctly is at most* $|\mathcal{H}|/2^p$*.*

Lemma 10. *Fix* $L \in \mathbb{N}$ *and an algorithm* A *that can make no more than* $\mathsf{q} = 2^{L-2}$ *oracle queries. Let* ic *be an ideal cipher uniformly chosen from the set* $\mathbb{IC}$ *with domain* $\mathcal{K} \times \{0,1\}^L$ *and image* $\{0,1\}^L$*.*[31] A *on input a hint* $h \in \mathcal{H}$ *adaptively makes forward/inverse queries to* ic*, and outputs* $p \leq 2^{L-2}$ *ideal primitive entries (as well as guesses for each of the entry values) that were not queried before. The probability (over the choice of* ic *and randomness of* A*) that there exists an* $h \in \mathcal{H}$ *where* $\mathsf{A}(h)$ *guesses all permutation entries correctly is at most* $|\mathcal{H}|/2^{p(L-1)}$*.*

Proof. The proof is deferred to full version. □

References

1. Alwen, J., Blocki, J.: Efficiently computing data-independent memory-hard functions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 241–271. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_9
2. Alwen, J., Blocki, J., Harsha, B.: Practical graphs for optimal side-channel resistant memory-hard functions. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.), ACM CCS 17, pp. 1001–1017. ACM Press, October/November 2017
3. Alwen, J., Blocki, J., Pietrzak, K.: Depth-robust graphs and their cumulative memory complexity. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 3–32. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_1
4. Alwen, J., Blocki, J., Pietrzak, K.: Sustained space complexity. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 99–130. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_4
5. Alwen, J., Chen, B., Kamath, C., Kolmogorov, V., Pietrzak, K., Tessaro, S.: On the complexity of scrypt and proofs of space in the parallel random oracle model. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 358–387. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_13

[31] A random permutation can be viewed as an ideal cipher with a fixed key.

6. Alwen, J., Chen, B., Pietrzak, K., Reyzin, L., Tessaro, S.: `Scrypt` is maximally memory-hard. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 33–62. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_2
7. Alwen, J., Serbinenko, V.: High parallel complexity graphs and memory-hard functions. In: Servedio, R.A. Rubinfeld, R. (eds.), 47th ACM STOC, pp. 595–603. ACM Press, June 2015
8. Alwen, J., Tackmann, B.: Moderately hard functions: definition, instantiations, and applications. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 493–526. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_17
9. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Ashby, V. (ed.), ACM CCS 93, pp. 62–73. ACM Press, November 1993
10. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2 password hash. Version 1.3 (2016). https://www.cryptolux.org/images/0/0d/Argon2.pdf
11. Blocki, J., Harsha, B., Kang, S., Lee, S., Xing, L., Zhou, S.: Data-independent memory hard functions: new attacks and stronger constructions. Cryptology ePrint Archive, Report 2018/944 (2018). http://eprint.iacr.org/2018/944
12. Blocki, J., Zhou, S.: On the depth-robustness and cumulative pebbling cost of Argon2i. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 445–465. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_15
13. Boneh, D., Corrigan-Gibbs, H., Schechter, S.: Balloon hashing: a memory-hard function providing provable protection against sequential attacks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 220–248. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_8
14. Dryja, T., Liu, Q.C., Park, S.: Static-memory-hard functions, and modeling the cost of space vs. time. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018, Part I. LNCS, vol. 11239, pp. 33–66. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03807-6_2
15. Dwork, C., Naor, M., Wee, H.: Pebbling and proofs of work. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 37–54. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_3
16. Dziembowski, S., Kazana, T., Wichs, D.: One-time computable self-erasing functions. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 125–143. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_9
17. Forler, C., Lucks, S., Wenzel, J.: Catena: a memory-consuming password scrambler. IACR Cryptology ePrint Archive 2013/525 (2013)
18. Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 21–39. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_2
19. Percival, C.: Stronger key derivation via sequential memory-hard functions. In: BSDCan 2009 (2009)
20. Percival, C., Josefsson, S.: The scrypt password-based key derivation function (2012)
21. Ren, L., Devadas, S.: Bandwidth hard functions for ASIC resistance. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 466–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_16
22. Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with composition: limitations of the indifferentiability framework. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 487–506. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_27