Static-Memory-Hard Functions, and Modeling the Cost of Space vs. Time

Thaddeus Dryja[1], Quanquan C. Liu[2], and Sunoo Park[2]

[1] MIT Media Lab, 75 Amherst St, Cambridge, MA, USA
tdryja@MIT.EDU
[2] MIT CSAIL, 32 Vassar St, Cambridge, MA, USA
{quanquan,sunoo}@mit.edu

Abstract. A series of recent research starting with (Alwen and Serbinenko, STOC 2015) has deepened our understanding of the notion of *memory-hardness* in cryptography — a useful property of hash functions for deterring large-scale password-cracking attacks — and has shown memory-hardness to have intricate connections with the theory of graph pebbling. Definitions of memory-hardness are not yet unified in the somewhat nascent field of memory-hardness, however, and the guarantees proven to date are with respect to a range of proposed definitions. In this paper, we observe *two* significant and practical considerations that are not analyzed by existing models of memory-hardness, and propose new models to capture them, accompanied by constructions based on new hard-to-pebble graphs. Our contribution is two-fold, as follows. First, existing measures of memory-hardness only account for *dynamic* memory usage (i.e., memory read/written at runtime), and do not consider *static* memory usage (e.g., memory on disk). Among other things, this means that memory requirements considered by prior models are inherently upper-bounded by a hash function's runtime; in contrast, counting static memory would potentially allow quantification of much larger memory requirements, decoupled from runtime. We propose a new definition of *static-memory-hard* function (SHF) which takes static memory into account: we model static memory usage by oracle access to a large preprocessed string, which may be considered part of the hash function description. Static memory requirements are *complementary* to dynamic memory requirements: neither can replace the other, and to deter large-scale password-cracking attacks, a hash function will benefit from being *both* dynamic-memory-hard and static-memory-hard. We give two SHF constructions based on pebbling. To prove static-memory-hardness, we define a new pebble game ("*black-magic pebble game*"), and new graph constructions with optimal complexity under our proposed measure. Moreover, we provide a prototype implementation of our first SHF construction (which is based on pebbling of a simple "cylinder" graph), providing an initial demonstration of practical feasibility for a limited range of parameter settings. Secondly, existing memory-hardness models implicitly assume that the cost of space and time are more or less on par: they consider only *linear* ratios between the costs of time and space. We propose a new model to capture *nonlinear* time-space trade-offs: e.g., how is the adversary impacted when space is quadratically more expensive than

time? We prove that nonlinear tradeoffs can in fact cause adversaries to employ different strategies from linear tradeoffs.
Please refer to the full version of our paper for all results, proofs, appendices, and implementation details [DLP18].

1 Introduction

Pebble games were originally formulated to model time-space tradeoffs by a game played on DAGs. Generally, a DAG can be thought to represent a computation graph where each node is associated with some computation and a pebble placed on a node represents saving the result of its computation in memory. Thus, the number of pebbles represents the amount of memory necessary to perform some set of computations. The natural complexity measures to optimize in this game is the minimum number of pebbles used, as well as the minimum amount of time it takes to finish pebbling all the nodes; these goals correspond with minimizing the amount of memory and time of computation.

Pebble games were first introduced to study programming languages and compiler construction [PH70] but have since then been used to study a much broader range of tasks such as register allocation [Set75], proof complexity [AdRNV17,Nor12], time-space tradeoffs in Turing machine computation [Coo73,HPV77], reversible computation [Ben89], circuit complexity [Pot17], and time-space tradeoffs in various algorithms such as FFT [Tom81], linear recursion [Cha73,SS79b], matrix multiplication [Tom81], and integer multiplication [SS79a] in the RAM as well as the external memory model [JWK81]. To see a more comprehensive survey of the results in pebbling up to the last couple of years, see [Pip82] up to the 1980s and [Nor15] up to 2015.

The relationship between pebbling and cryptography has been a subject of research interest for decades, which has enjoyed renewed activity in the last few years. A series of recent works [AB16,ABH17,ABP17a,ABP17b,AS15,AT17,ACP+16,AAC+17,BZ16,BZ17] has deepened our understanding of the notion of *memory-hardness* in cryptography, and has shown memory-hardness to have intricate connections with the theory of graph pebbling.

Memory-hard functions (MHFs) have garnered substantial recent interest as a security measure against adversaries trying to perform attacks at scale, particularly in the ubiquitous context of password hashing. Consider the following scenario: hashes of user passwords are stored in a database,and when a user enters a password p to log in, her computer sends $H(p)$ to the database server, and the server compares the received hash to its stored hash for that user's account. For a normal user, it would be no problem if hash evaluation were to take, say, one second. An attacker trying to guess the password by brute-force search, on the other hand, would try orders of magnitude more passwords, so a one-second hash evaluation could be prohibitively expensive for the attacker.

The evolution of password hashing functions has been something of an arms race for decades, starting with the ability to increase the number of rounds in the DES-based unix `crypt` function to increase its computation time—a feature that was used for exactly the above purpose of deterring large-scale

password-cracking. Attackers responded by building special-purpose circuits for more efficient evaluation of `crypt`, resulting in a gap between the evaluation cost for an attacker and the cost for an honest user.

A promising approach to mitigating this asymmetry in cost between hash evaluation on general- and special-purpose hardware is to increase the use of *memory* in the password hashing function. Memory is implemented in standardized ways which have been highly optimized, and memory chips are widely regarded to be an interchangeable commodity. Commonly used forms of memory — whether on-die SRAM cache, DRAM, or hard disks — are already optimized for the purpose of data I/O operations; and while there is active research in improving memory access times and costs, progress is and has been relatively incremental. This state of affairs sets up a relatively "even playing field," as the normal user and the attacker are likely to be using memory chips of similar memory access speed. While an attacker may choose to buy more memory, the cost of doing so scales linearly with the amount purchased.

The designs of several MHFs proposed to date (e.g., [Per09,AS15,AB16,ACP+16,ABP17a]) have proven memory-hardness guarantees by basing their hash function constructions on DAGs, and using space complexity bounds from graph pebbling. Definitions of memory-hardness are not yet unified in this somewhat nascent field, however — the first MHF candidate was proposed only in 2009 [Per09] — and the guarantees proven are with respect to a range of definitions. The "cumulative complexity"-based definitions of [AS15] have enjoyed notable popularity, but some of their shortcomings have been pointed out by subsequent work proposing alternative more expressive measures, in particular, [ABP17b,AT17].

Our contribution We observe *two* significant and practical considerations not analyzed by existing models of memory-hardness, and propose new models to capture them, accompanied by constructions based on new hard-to-pebble graphs. Our main contribution is two-fold, as described in (1) and (2) below. We also provide an additional contribution of separate interest, described in (3).

1. ***Static-memory-hardness.*** Existing measures of memory-hardness only account for *dynamic* memory usage (i.e., memory read/written at runtime), and do not consider *static* memory usage (e.g., memory on disk). Among other things, this means that memory requirements considered by prior models are inherently upper-bounded by a hash function's runtime; in contrast, counting static memory would potentially allow quantification of much larger memory requirements, decoupled from the honest evaluator's runtime.
   We propose a new definition of *static-memory-hard* function (SHF) (Definition 24), and present two SHF constructions based on pebbling. To prove static-memory-hardness, we define a new pebble game called the *black-magic pebble game* (Definition 2), and prove properties of the space complexity of this game for new graphs (Graph Constructions 2 and 8). Graph Construction 8 gives rise to an SHF with a better asymptotic guarantee (same space usage but sustained over more time), whereas Graph Construction 2 yields an SHF with the advantage of simplicity in practice. Informal theorems stating the constructions' static-memory-hardness guarantees are given in

Section 1.2 and formal theorems are in Section 5. In our full version [DLP18], we discuss our prototype implementation based on Graph Construction 2. We emphasize that static memory requirements are *complementary* to dynamic memory requirements: neither can replace the other, and to deter large-scale password-cracking attacks, a hash function will benefit from being *both* dynamic-memory-hard and static-memory-hard.

2. ***Modeling nonlinear cost of space vs. time.*** Existing measures of memory-hardness implicitly assume a linear trade-off between the costs of space and time. This model precludes situations where the relative costs of space and time might be more unbalanced (e.g., quadratic or cubic). We demonstrate that this modeling limitation is significant, by giving an example where adversaries facing asymptotically different space-time cost tradeoffs would in fact employ *different strategies*. Then, to remedy this shortcoming, we define *graph-optimal* variants of memory-hardness measures (in Section 2) that *explicitly* model the relative cost of space and time. These can be seen as extending the main memory-hardness measures in the literature (namely, *cumulative complexity* and *sustained memory complexity*). We prove bounds on the new measure as elaborated in Section 1.2.
3. We give the first graph construction that is tight, up to $\log\log n$-factors, to the optimal cumulative complexity that can be achieved for any graph (upper bound due to [ABP17a,ABP17b]).

   Informal version of Theorem 6.23 in [DLP18]. There exists a family of graphs where the cumulative complexity of any constant in-degree graph with n nodes in the family is $\Theta\left(\frac{n^2 \log\log n}{\log n}\right)$ which is asymptotically tight to the upper bound of $\Theta\left(\frac{n^2 \log\log n}{\log n}\right)$ given in [ABP17a,ABP17b] in the sequential pebbling model.

The full version [DLP18] gives a brief background on graph pebbling, Section 1.1 gives discussion on memory-hardness measures and related work, and Sections 1.2 and 1.2 give more detailed high-level overviews of our SHF contribution and nonlinear space-time tradeoff model (items (1) and (2) above), respectively.

Graph pebbling and memory-hardness Graph pebbling algorithms can be used to construct hash functions in the (parallel) random oracle model. This paradigm has been used by prior constructions of memory-hard hashing [AS15] as well as other prior works [DKW11].

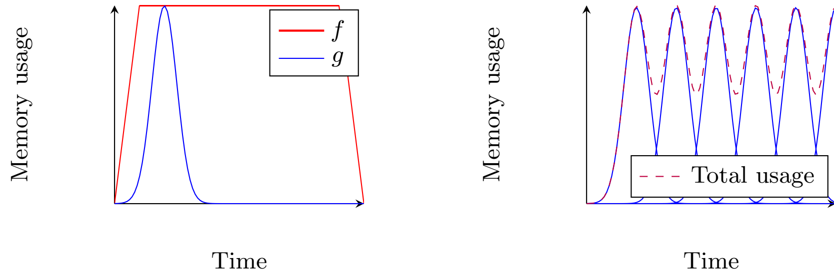
Informally, the idea to "convert" a graph into a hash function is to associate with each node v a string called a *label*, which is defined to be $\mathcal{O}(v, \mathsf{pred}(v))$ where $\mathcal{O}$ is a random oracle and $\mathsf{pred}(v)$ is the list of labels of predecessors of v. For source nodes, the label is instead defined to be $\mathcal{O}(v, \zeta)$ for a string ζ which is an input to the hash function. The output of the hash function is defined to be the list of labels of target nodes. Intuitively, since the label of a node cannot be computed without the "random" labels of all its predecessors, any algorithm computing this hash function must move through the nodes of the graph according to rules very similar to those prescribed by the pebbling

game; and therefore, the memory requirement of computing the hash function roughly corresponds to the pebble requirement of the graph. Thus, proving lower bounds on the pebbling complexity of graph families has useful implications for constructing provably memory-hard functions.

In our setting, in contrast to previous work, we employ a variant of the above technique: the string ζ is a fixed parameter of our hash function, and the input to the hash function instead specifies the indices of the target nodes whose labels are to be outputted.

1.1 Discussion on memory-hardness measures and related work

The original paper proposing memory-hard functions [Per09] suggested a very simple measure: the minimum amount of memory necessary to compute the hash function. It was subsequently observed that a major drawback of this measure is that it does not distinguish between functions f and g with the same peak memory usage, even if the peak memory lasts a long time in evaluating f and is just fleeting in evaluating g (Figure 1a). This is significant as the latter type of function is much better for a password-cracking adversary. In particular, pipelining the evaluation of the latter type of function would allow reuse of the same memory for many function evaluations at once, effectively reducing the adversary's amortized memory requirement by a factor of the number of concurrent executions (Figure 1b).



(a) Functions with the same peak memory usage

(b) Pipelined evaluations of g (reusing memory)

Fig. 1: Limitations of peak memory usage as a memory-hardness measure

Cumulative complexity [AS15] put forward the notion of *cumulative complexity* (CC), a complexity measure on graphs. CC was adopted by several subsequent works as a canonical measure of memory-hardness. CC measures the *cumulative* memory usage of a graph pebbling function evaluation: that is, the sum of memory usage over all time-steps of computation. In other words, this is the area under a

graph of memory usage against time. CC is designed to be very robust against amortization, and in particular, scales linearly when computing many copies of a function on different inputs. This is a great advantage compared to the simpler measure of [Per09], which does not account well for an amortizing adversary (as shown in Figure 1).

Depth-robust graphs More recently, [AB16,ABP17a] proved bounds on optimal CC of certain graph families. They showed that a particular graph property called *depth-robustness* suffices to attain optimal CC (up to polylog factors–the CC of any graph with bounded in-degree is upper bounded by $O\left(\frac{n^2 \log\log n}{\log n}\right)$ [AB16,ABP17b]). An (r, s)-depth-robust graph is one where there exists a path of length s even when any r vertices are removed. Intuitively, this captures the notion that storing any r vertices of the graph will not shortcut the pebbling in a significant way. It turns out that depth-robustness will again be a useful property in our new model of memory-hardness with preprocessing.

Sustained memory complexity Very recently, Alwen, Blocki, and Pietrzak [ABP17a] proposed a new measure of memory complexity, which captures not only the cumulative memory usage over time (as does CC), but goes further and captures the amount of time for which a particular level of memory usage is sustained. Our SHF definition also captures *sustained* memory usage: we propose a definition of capturing the duration for which a given amount of memory is required, designed to capture static as well as dynamic memory requirements. By the nature of static memory, it is especially appropriate in our setting to consider (and maximize) the amount of time for which a static memory requirement is *sustained*.

Core-area memory ratio Previous works have considered certain hardware-dependent non-linearities in the ratio between the cost of memory and computation [BK15,AB16,RD17]. Such phenomena may incur a multiplicative factor increase in the memory cost that is dependent, in a possibly non-linear way, on specific hardware features. Note that *the non-linearity here is in the hardware-dependence*, rather than the space-time tradeoff itself. In contrast, our new models are more expressive, in that they make configurable the asymptotic tradeoff between space and time (by a parameter α which is in the exponent, as detailed in Definition 16) in an application-dependent way. This versatility of configuration targets applications where the trade-off may realistically depend on arbitrary and possibly exogenous space/time costs, and thus contrasts with metrics tailored for a specific hardware feature, such as core-memory ratio.

Towards a general theory of moderately hard functions Most recently, Alwen and Tackmann [AT17] proposed a more general (though not comprehensive) framework for defining desirable guarantees of “moderately hard functions,” i.e., functions that are efficient to compute but somewhat hard to invert. Their work points out a number of drawbacks of prior measures such as those described above. Notably, many of the prior measures characterized the hardness of *computing* the function with an implicit assumption that this hardness would translate to the hardness of *inverting* the function (as it would indeed in the case of a brute-force approach to inversion). In other words, these measures implicitly assume that

the hash function in question “behaves like a random oracle” in the sense that brute-force inversion is the optimal approach.

1.2 Our contributions in more detail

To prove static-memory-hardness, we define a new pebble game called the *black-magic pebble game* (Definition 2), and prove properties of the space complexity of this game for new graphs (Graph Constructions 2 and 8).

The black-magic pebble game may additionally be of independent interest for the pebbling literature. Indeed, a pebble game used to analyze security of *proofs of space* [DFKP15] can be viewed as a non-adaptiveversion of the black-magic pebble game in which the target node set is sampled from a distribution by a challenger.

Based on our new graph constructions, we construct SHFs with provable guarantees on sustained memory usage, as follows. Graph Construction 8 gives a better asymptotic guarantee (same space usage but sustained over more time), whereas Graph Construction 2 has the advantage of simplicity in practice. In our full version [DLP18], we discuss our prototype implementation based on Graph Construction 2.

Static-memory-hard functions (SHFs) Prior memory-hardness measures make a modeling assumption: namely, that the memory usage of interest is solely that of memory dynamically generated at run-time. However, static memory can be costly for the adversary too, and yet it is not taken into account by existing measures such as CC. Intuitively, it can be beneficial to design a function whose evaluation requires keeping a large amount of static memory on disk (which may be thought to be produced in a one-time initial setup phase). While not all the static memory might be accessed in any given evaluation, the “necessity” to maintain the data on disk can arise from the idea that an adversary attempting to evaluate the function on an arbitrary input while having stored a lesser amount of data would be forced to *dynamically* generate comparable amounts of memory. Note that the resulting *dynamic* memory requirements could be orders of magnitude larger (say, gigabytes) than the memory requirements of existing memory-hard function proposals, because unlike in prior memory-hardness models, here we have decoupled the memory requirement from the memory requirements of the honest evaluator.

We propose a new model and definitions for *static-memory-hard functions* (SHFs), in which we model static memory usage by oracle access to a large preprocessed string, which may be considered part of the hash function description. In particular, the preprocessed string can be public and known to the adversary — the important guarantee is that without storing (almost) all of it statically, the adversary will incur huge online memory requirements.

Definition (informal). We model a *static-memory-hard function family* as a two-part algorithm $\mathcal{H} = (\mathcal{H}_1, \mathcal{H}_2)$ in the parallel random oracle model, where $\mathcal{H}_1(1^\kappa)$ outputs a “large” string to which $\mathcal{H}_2$ has oracle access,and $\mathcal{H}_2$ receives an input x and outputs a hash function output y. Informally, our hardness

requirement is that with high probability, any *two-part* adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ must *either* have $\mathcal{A}_1$ output a large state (comparable to the output size of $\mathcal{H}_1$), *or* have $\mathcal{A}_2$ use large (dynamic) space.

We then give two constructions of SHFs based on graph pebbling. To prove static-memory-hardness, we define a new pebble game called the *black-magic pebble game* of which we give an overview in Section 1.2. Our simpler SHF construction is based on a family of tree-like "cylinder" graphs, which achieves memory usage proportional to the square root of the number of nodes, sustained over time proportional to the square root of the number of nodes. Furthermore, we give a better construction based on pebbling of a new graph family, that achieves better parameters: the same (square root) memory usage, but sustained over time proportional to the number of nodes.

Informal version of Theorem 13. The "cylinder graph" (Graph Construction 2) can be used to construct an SHF with static memory requirement $\Lambda \in \Theta(\sqrt{n}/(\kappa - \xi \log(\kappa))$ where n is the number of nodes in the graph, κ is a security parameter, and $\xi \in \omega(1)$, such that any adversary using non-trivially less *static* memory than Λ must incur at least Λ *dynamic* memory usage for at least $\Theta(\sqrt{n})$ steps.

Informal version of Theorem 14. Graph Construction 8 can be used to construct an SHF with static memory requirement $\Lambda \in \Theta(\sqrt{n})/(\kappa - \xi \log(\kappa))$ where n, κ, and ξ are as described above, such that any adversary using non-trivially less *static* memory than Λ must incur at least Λ *dynamic* memory usage for at least $\Theta(n)$ steps.

Static memory requirements are *complementary* to dynamic memory requirements: neither can replace the other, and to deter large-scale password-cracking attacks, a hash function will benefit from being *both* dynamic-memory-hard and static-memory-hard. In Section 4.1, we give a discussion of how, given a static-memory-hard function and a (dynamic-)memory-hard function, they can be concatenated to yield a "dynamic SHF" that inherits both the static memory requirement of the former and the dynamic memory requirement of the latter.

Black-magic pebble game We introduce a new pebble game called the *black-magic pebble game*. This game bears some similarity to the standard (black) pebble game, with the main difference that the player has access to an additional set of pebbles called *magic pebbles*. Magic pebbles are subject to different rules from standard pebbles: they may be placed anywhere at any time, but cannot be removed once placed, and may be limited in supply. The pebbling space cost of this game is defined as the maximum number of standard pebbles on the graph at any time-step plus the total number of magic pebbles used throughout the computation. Observe that while the most time-efficient strategy in the black-magic pebble game is always to pebble all the target nodes with magic pebbles in the first step, the most space-efficient strategy is much less clear.

Lower-bounds on space usage can be non-trivially different between the standard and magic pebbling games. For example, if a graph has a constant

number of targets, then magic pebbling space usage will never exceed a constant number of pebbles, whereas the standard pebbling space usage can be super-constant. In particular, it is unclear, in the new setting of magic pebbling, whether known lower-bounds on pebbling space usage in the standard pebble gameare transferable to the magic pebble game. We prove in Section 5 that for layered graphs,the best possible lower-bound for the magic pebbling game is $\Theta(\sqrt{n})$.

We leave determining the lower bound for magic pebbling space usage in general graphs as an open question. An answer to this open question would be useful towards constructing better static-memory-hard functions using the paradigm presented herein.

Our proof techniques rely on a close relationship between black-magic pebbling complexity and a new graph property which we define, called *local hardness*. Local hardness considers black-magic pebbling complexity in a variant model where *subsets* of target nodes are required to be pebbled (rather than *all* target nodes, as in the traditional pebbling game), and moreover, a "preprocessing phase" is allowed, wherein magic pebbles may be placed on the graph in advance of knowing which target nodes are to be produced. This "preprocessing" aspect bears some resemblance to the *black-white pebbling game* [CS74], a variant of the standard pebbling game in which some limited number of *white* pebbles can be placed "for free," and the black pebbles must be placed according the standard rules. However, our setting differs from the black-white pebbling game: while preprocessing and storing magic pebbles in advance can be viewed as analogous to placing white pebbles for free, the black-white pebbling game imposes restrictions on the *removal* of white pebbles from the graph, which are not present in our setting.

Capturing relative cost of memory vs. time Existing measures such as CC and sustained memory complexity trade off space against time at a linear ratio. In particular, CC measures the minimal area under a graph of memory usage against time, over all possible algorithms that evaluate a function.

However, different applications may have different relative cost of space and time. We propose and define a variant of CC called α-CC, parametrized by α which determines the relative cost of space and time, and observe that α-CC may be meaningfully different from CC and more suitable for certain application scenarios. For example, when memory is "quadratically" more expensive than time, the measure of interest to an adversary may be the area under a graph of memory squared against time, as demonstrated by the following theorem.

Informal version of Theorem 6.8 in [DLP18]. There exist graphs for which an adversary facing a linear space-time cost trade-off would in fact employ a *different pebbling strategy* from one facing a cubic trade-off.

It follows that when the costs of space and time are not linearly related, the CC measure may be measuring the complexity of *the wrong algorithm*, i.e., not the algorithm that an adversary would in fact favor. We thus see that our α-CC measure is more appropriate in settings where space may be substantially more

costly than time (or vice versa). Moreover, our parametrized approach generalizes naturally to sustained memory complexity. We show that our graph constructions are invariant across different values of α, a potentially desirable property for hash functions so that they are robust against different types of adversaries.

Informal version of Theorem 6.13 in [DLP18]. Given any graph construction $G = (V, E)$, there exists a pebbling strategy that is less expensive asymptotically than any strategy using a number of pebbles asymptotically equal to the number of nodes in the graph for any time-space tradeoff.

Please refer to the full version of our paper for all results, proofs, appendices, and implementation details [DLP18].

2 Pebbling definitions

A *pebbling game* is a one-player game played on a DAG where the goal of the player is to place pebbles on a set of one or more *target nodes* in the DAG.

In Section 2.1, we formally define two variations of the sequential and parallel pebble games: the *standard (black) pebble game* and the *black-magic pebble game*, the latter of which we introduce in this work. We also give the definitions of valid strategies and moves in these games. Then in Section 2.2, we define measures for evaluating the sequential and parallel pebbling complexity on families of graphs.

2.1 Standard and magic pebbling definitions

Definition 1 (Standard (black) pebble game).

- ***Input:*** *A DAG,* $G = (V, E)$*, and a* target set $T \subseteq V$*. Define* $\mathsf{pred}(v) = \{u \in V : (u, v) \in E\}$*, and let* $S \subseteq V$ *be the set of sources of* G*.*
- ***Rules at move*** i***:*** *At the start of the game, no node of* G *contains a pebble. The player has access to a supply of* black pebbles*. Game-play proceeds in discrete moves, and* P_i *(called a "pebble configuration") is defined as the set of nodes containing pebbles after the* i*th move.* $P_0 = \varnothing$ *represents the initial configuration where no pebbles have been placed. Each move may consist of multiple actions adhering to the following rules.*
  1. *A pebble can be placed on any source,* $s \in S$*.*
  2. *A pebble can be removed from any vertex.*
  3. *A pebble can be placed on a non-source vertex,* v*, if and only if its direct predecessors were pebbled at time* $i - 1$ *(i.e.,* $\mathsf{pred}(v) \in P_{i-1}$*).*
  4. *A pebble can be moved from vertex* v *to vertex* w *if and only if* $(v, w) \in E$ *and* $\mathsf{pred}(w) \in P_{i-1}$*.*
- ***Goal:*** *Pebble all nodes in* T *at least once (i.e.,* $T \subseteq \bigcup_{i=0}^{t} P_i$*).*

Remark 1. At first glance, it may seem that rule 4 in Definition 1 is redundant as a similar effect can be achieved by a combination of the other rules. However, the application of rule 4 can allow the usage of fewer pebbles. For example, a simple

two-layer binary tree (with three nodes) could be pebbled with two pebbles using rule 4, but would require three pebbles otherwise. Nordstrom [Nor15] showed that in sequential strategies, it is always possible to use one fewer pebble by using rule 4.

We note for completeness that while rule 4 is standard in the pebbling literature, not all the papers in the MHF literature include rule 4.

Next, we define the *black-magic pebble game* which we will use to prove security properties of our static-memory-hard functions.

Definition 2 (Black-magic pebble game).

- ***Input:*** *A DAG $G = (V, E)$, a target set $T \subseteq V$, and* magic pebble bound $\mathfrak{M} \in \mathbb{N} \cup \{\infty\}$.
- ***Rules:*** *At the start of the game, no node of G contains a pebble. The player has access to two types of pebbles:* black pebbles *and up to $\mathfrak{M}$* magic pebbles*. Game-play proceeds in discrete moves, and $P_i = (M_i, B_i)$ is the pebble configuration after the ith move, where M_i, B_i are the sets of nodes containing magic and black pebbles after the ith move, respectively. $P_0 = (\varnothing, \varnothing)$ represents the initial configuration where no black pebbles or magic pebbles have been placed. Each move may consist of multiple actions adhering to the following rules.*
  1. *Black pebbles can be placed and removed according to the rules of the standard pebble game which are defined in the full version.*
  2. *A magic pebble can be placed on and removed from any node, subject to the constraint that at most $\mathfrak{M}$ magic pebbles are used throughout the game.*
  3. *Each magic pebble can be placed at most once: after a magic pebble is removed from a node, it disappears and can never be used again.*
- ***Goal:*** *Pebble all nodes in T at least once (i.e., $T \subseteq \bigcup_{i=0}^{t} (M_i \cup B_i)$).*

Remark 2. In the black-magic pebble game, unlike in the standard pebble game, there is always the simple strategy of placing magic pebbles directly on all the target nodes. At first glance, this may seem to trivialize the black-magic game. When optimizing for space usage, however, this simple strategy may not be favorable for the player: by employing a different strategy, the player might be able to use much fewer than T pebbles overall.

Next, we define valid sequential and parallel strategies in these games.

Definition 3 (Pebbling strategy). *Let G be a graph and T be a target set. A* standard (resp., black-magic) pebbling strategy for (G, T) *is defined as a sequence of pebble configurations, $\mathcal{P} = \{P_0, \ldots, P_t\}$, satisfying conditions 1 and 2 below. $\mathcal{P}$ is moreover* valid *if it satisfies condition 3, and* sequential *if it satisfies condition 4.*

1. *$P_0 = \varnothing$.*
2. *For each $i \in [t]$, P_i can be obtained from P_{i-1} by a legal move in the standard (resp., black-magic) pebble game.*

3. *$\mathcal{P}$ successfully pebbles all targets, i.e.,* $T \subseteq \bigcup_{i=0}^{t} P_i$.
4. *For each $i \in [t]$, P_i contains at most one vertex not contained in P_{i-1} (i.e., $|P_i \setminus P_{i-1}| \leq 1$).*

A black-magic pebbling strategy must satisfy one additional condition to be considered valid*:*

5. *At most $\mathfrak{M}$ magic pebbles are used throughout the strategy, i.e.,* $|\bigcup_{i\in[t]} M_i| \leq \mathfrak{M}$ *where M_i is the ith configuration of magic pebbles.*

2.2 Cost of pebbling

In this subsection, we give definitions of several cost measures of graph pebbling, applicable to the standard and black-magic pebbling games. While these definitions assume parallel strategies, we note that the sequential versions of the definitions are entirely analogous.

Space complexity in standard pebbling We give a brief informal summary of the definitions in this subsection, before proceeding to the formal definitions.

Pebbling complexity measures We informally overview the pebbling complexity definitions, some of which are new to this work.

The *time complexity* of a pebbling strategy $\mathcal{P}$ is the number of steps, i.e., $\mathsf{Time}(\mathcal{P}) = |\mathcal{P}|$. The *time complexity* of a graph $G = (V, E)$ given that at most S pebbles can be used is $\mathsf{Time}(G, S) = \min_{\mathcal{P}\in\mathbb{P}_{G,T,S}} (\mathsf{Time}(\mathcal{P}))$. Next, we overview variants of space complexity.

1. **Space complexity** of a *pebbling strategy* $\mathcal{P}$ on a graph G, denoted by $\mathbf{P}_{\mathrm{s}}(\mathcal{P})$, is the minimum number of pebbles required to execute $\mathcal{P}$. Space complexity of the *graph* G with target set T, written $\mathbf{P}_{\mathrm{s}}(G, T)$, is the minimum space complexity of any valid pebbling strategy for G.
2. **Λ-sustained space complexity** [ABP17a]of a *pebbling strategy* $\mathcal{P}$ on a graph G, denoted by $\mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \Lambda)$, is the number of time-steps during the execution of $\mathcal{P}$, in which at least Λ pebbles are used. Λ-sustained space complexity of the *graph* G with target set T, written $\mathbf{P}_{\mathrm{ss}}(G, \Lambda, T)$ is the minimum Λ-sustained space complexity of all valid pebbling strategies for G.
3. **Graph-optimal sustained complexity** of a *pebbling strategy* $\mathcal{P}$, denoted by $\mathbf{P}_{\text{opt-ss}}(\mathcal{P})$, is the number of time-steps during the execution of $\mathcal{P}$, in which the number of pebbles in use is equal to the space complexity of G. Graph-optimal sustained complexity of the *graph* G with target set T, written $\mathbf{P}_{\text{opt-ss}}(G, T)$ is the minimum graph-optimal sustained complexity of all valid pebbling strategies for G.
4. **Δ-suboptimal sustained complexity** of a *pebbling strategy* $\mathcal{P}$ is the number of time-steps, during the execution of $\mathcal{P}$, in which the number of pebbles in use is at least the space complexity of G *minus* Δ. Δ-suboptimal sustained complexity of the *graph* G is the minimum Δ-suboptimal sustained complexity of all valid pebbling strategies for G.

A couple of remarks are in order.

Remark 3. The third and fourth definitions are new to this paper. They can be seen as special variants of Λ-sustained space complexity, i.e., with a special setting of Λ dependent on the specific graph family in question. They are useful to define in their own right, as unlike plain Λ-sustained space complexity, these measures express complexity for a given graph family relative to the best possible value of Λ at which sustained space usage could be hoped for. In the rest of this paper, we prove guarantees on *graph-optimal sustained complexity* of our constructions, which have high sustained space usage at the optimal Λ-value. However, we also define *Δ-suboptimal sustained complexity* here for completeness, since it is more generaland preferable to graph-optimal complexity when evaluating graph families where the maximal space usage may not be sustained for very long.

Remark 4. We have found the term "Λ-sustained space complexity" can be slightly confusing, in that it measures a number of time-steps rather than an amount of space. We retain the original terminology as it was introduced, but include this remark to clarify this point.

We now present the formal definitions of the complexity measures for the standard pebbling game. In all of the below definitions, $G = (V, E)$ is a graph, $T \subseteq V$ is a target set, $\mathcal{P} = (P_1, \ldots, P_t)$ is a standard pebbling strategy on (G, T), and $\mathbb{P}_{G,T}$ denotes the set of all valid standard pebbling strategies on (G, T).

Definition 4. *The* space complexity of pebbling strategy $\mathcal{P}$ *is:* $\mathbf{P}_{\mathrm{s}}(\mathcal{P}) = \max_{P_i \in \mathcal{P}} (|P_i|)$. *The* space complexity of G *is the minimal space complexity of any valid pebbling strategy that pebbles the target set $T \subset V$:* $\mathbf{P}_{\mathrm{s}}(G, T) = \min_{\mathcal{P}' \in \mathbb{P}_{G,T}} (\mathbf{P}_{\mathrm{s}}(\mathcal{P}'))$.

Definition 5. *The* Λ-sustained space complexity of $\mathcal{P}$ *is:* $\mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \Lambda) = |\{P_i : |P_i| \geq \Lambda\}|$. *The* Λ-sustained space complexity of G *is the minimal Λ-sustained space complexity of any valid pebbling strategy that pebbles the target set $T \subseteq V$:* $\mathbf{P}_{\mathrm{ss}}(G, \Lambda, T) = \min_{\mathcal{P}' \in \mathbb{P}_{G,T}} (\mathbf{P}_{\mathrm{ss}}(\mathcal{P}', \Lambda))$.

Definition 6. *The* graph-optimal sustained complexity of $\mathcal{P}$ *is:* $\mathbf{P}_{\text{opt-ss}}(\mathcal{P}) = \mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \mathbf{P}_{\mathrm{s}}(G, T))$. *The* graph-optimal sustained complexity of G *is the minimal graph-optimal sustained complexity of any valid pebbling strategy that pebbles the target set $T \subseteq V$:* $\mathbf{P}_{\text{opt-ss}}(G, T) = \min_{\mathcal{P}' \in \mathbb{P}_{G,T}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}'))$.

Definition 7. *The* Δ-suboptimal sustained complexity of $\mathcal{P}$ *is:*

$$\mathbf{P}_{\text{opt-ss}}(\mathcal{P}, \Delta) = \mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \mathbf{P}_{\mathrm{s}}(G, T) - \Delta).$$

The Δ-suboptimal sustained complexity of G *is the minimal graph-optimal sustained complexity of any valid pebbling strategy that pebbles the target set $T \subseteq V$:* $\mathbf{P}_{\text{opt-ss}}(G, \Delta, T) = \min_{\mathcal{P}' \in \mathbb{P}_{G,T}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}', \Delta))$.

Time complexity in standard pebbling We present the following formal definitions for measuring the time complexity of strategies in the standard pebble game. In all the below definitions, $G = (V, E)$ is a graph, $T \subseteq V$ is a target set, $\mathcal{P} = (P_1, \ldots, P_t)$ is a standard pebbling strategy on (G, T) where $\mathbb{P}_{G,T,S}$ denotes the set of all valid pebbling strategies on (G, T) that use at most S pebbles.

Definition 8. *The* time complexity *of a pebbling strategy* $\mathcal{P}$ *is* $\mathsf{Time}(\mathcal{P}) = |\mathcal{P}|$. *The* time complexity *of a graph* $G = (V, E)$ *given that at most* S *pebbles can be used is* $\mathsf{Time}(G, S) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,S}} (\mathsf{Time}(\mathcal{P}))$.

Space complexity in black-magic pebbling Next, we define the corresponding complexity notions for the black-magic pebbling game. As above, $G = (V, E)$ is a graph, $T \subseteq V$ is a target set, and $\mathfrak{M}$ is a magic pebble bound. In this subsection, $\mathcal{P} = (P_1, \ldots, P_t) = ((M_1, B_1), \ldots, (M_t, B_t))$ denotes a black-magic pebbling strategy on (G, T). Moreover, $\mathbb{M}_{G,T,\mathfrak{M}}$ denotes the set of all valid magic pebbling strategies on (G, T), and $m(\mathcal{P})$ denotes the total number of magic pebbles used in the execution of $\mathcal{P}$.

Definition 9. *The* (magic) space complexity *of* $\mathcal{P}$ *is:* $\mathbf{P}_{\mathrm{s}}(\mathcal{P}) = \max(m(\mathcal{P}), \max_{P_i \in \mathcal{P}}(|P_i|))$. *The* (magic) space complexity of G w.r.t. $\mathfrak{M}$ *is the minimal space complexity of any valid magic pebbling strategy that pebbles the target set* $T \subseteq V$*:* $\mathbf{P}_{\mathrm{s}}(G, \mathfrak{M}, T) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,\mathfrak{M}}} (\mathbf{P}_{\mathrm{s}}(\mathcal{P}))$.

Remark 5. We briefly provide some intuition for the complexity measure defined above in Def. 9. If we consider all magic pebbles to be static memory objects that were saved from a previous evaluation of the hash function, then the total number of magic pebbles is the amount of memory that was used to save the results of a previous evaluation of the hash function. Because of this, it is natural to take the maximum of the memory used to store results from a previous evaluation of the function and the current memory that is used by our current pebbling strategy since that would represent how much memory was used to compute the results of hash function during the current evaluation.

Definition 10. *The* (magic) Λ-sustained space complexity *of* $\mathcal{P}$ *is:* $\mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \Lambda) = |\{P_i : |P_i| \geq \Lambda\}|$. *The* Λ-sustained space complexity of G w.r.t. $\mathfrak{M}$ and $T \subseteq V$ *is:* $\mathbf{P}_{\mathrm{ss}}(G, \Lambda, \mathfrak{M}, T) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,\mathfrak{M}}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}, \Lambda))$.

Definition 11. *The* (magic) graph-optimal sustained complexity *of* $\mathcal{P}$ *is:* $\mathbf{P}_{\text{opt-ss}}(\mathcal{P}) = \mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \mathbf{P}_{\mathrm{s}}(G, T))$. *The* graph-optimal sustained complexity of G w.r.t. $\mathfrak{M}$ and $T \subseteq V$ *is:* $\mathbf{P}_{\text{opt-ss}}(G, \mathfrak{M}, T) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,\mathfrak{M}}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}))$.

Definition 12. *The* (magic) Δ-suboptimal sustained complexity *of* $\mathcal{P}$ *is:* $\mathbf{P}_{\text{opt-ss}}(\mathcal{P}, \Delta) = \mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \mathbf{P}_{\mathrm{s}}(G, T) - \Delta)$. *The* Δ-suboptimal sustained complexity of G w.r.t. $\mathfrak{M}$ and $T \subseteq V$ *is:*

$$\mathbf{P}_{\text{opt-ss}}(G, \Delta, \mathfrak{M}, T) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,\mathfrak{M}}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}, \Delta)).$$

2.3 Incrementally hard graphs

We introduce the following definition for our notion of graphs which require $|T|$ pebbles to pebble regardless of the number of targets that are asked, given a constraint on the number of magic pebbles that can be used. This concept has not been previously analyzed in the pebbling literature; traditional pebbling complexity usually treats graphs with fixed target sets.

Definition 13 (Incremental Hardness). *Given at most $\mathfrak{M}$ magic pebbles, for any subset of targets $C \subseteq T$ where $|C| > \mathfrak{M}$, the number of pebbles (magic and black pebbles) necessary in the black-magic pebble game to pebble C is at least $|T|$ where the number of magic pebbles used in this game is upper bounded by $\mathfrak{M}$: $\mathbf{P}_{\mathrm{s}}(G, |C|-1, C) \geq |T|$.*

α-tradeoff cumulative complexity *α-tradeoff cumulative complexity*, or CC^{α}, is a new measure introduced in this paper, which accounts for situations where space and time do not trade off linearly. Similar notions to this have been explored before e.g. [FLW13], [BK15,AB16,RD17]. A discuss of the *core-area memory ratio* [BK15,AB16,RD17] can be found in Section 1.1. They considered the notion of λ-memory-hardness where intuitively $S \cdot T = \Omega\left(G^{\lambda+1}\right)$ where the space-time cost is some exponential of the size of the stored graph [FLW13]. We note that this notion is very different from our notion of α-tradeoff complexity since they only consider the space-time cost (not cumulative complexity) and do not consider nonlinear tradeoffs between space and time (one can just consider $G^{\lambda+1}$ to a constant in the tradeoff curve).

Here, we see the usefulness of defining sustained complexities in terms of the minimum required space (as opposed to being parametrized by Λ) since we can always obtain an upper bound on CC^{α}, for *any* α, of a graph directly from our proofs of the space complexity and sustained time complexity of a DAG.

Definition 14 (Standard pebbling α-space cumulative complexity). *Given a valid parallel standard pebbling strategy, $\mathcal{P}$, for pebbling a graph $G = (V, E)$, the* standard pebbling α-space cumulative complexity *is the following:*

$$\mathsf{p\text{-}cc}_{\alpha}(G, \mathcal{P}) = \sum_{P_i \in \mathcal{P}} |P_i|^{\alpha} \ .$$

Definition 15 (Black-magic pebbling α-space cumulative complexity). *Given a valid parallel black-magic pebbling strategy, $\mathcal{P}$, for pebbling a graph $G = (V, E)$, the* black-magic pebbling α-space cumulative complexity *is the following:*

$$\mathsf{p\text{-}cc}^{M}_{\alpha}(G, \mathcal{P}) = \max\left(m(\mathcal{P})^{\alpha}, \sum_{P_i \in \mathcal{P}} |P_i|^{\alpha}\right) = \max\left(m(\mathcal{P})^{\alpha}, \sum_{P_i \in \mathcal{P}} |B_i \cup M_i|^{\alpha}\right)$$

where $m(\mathcal{P})$ denotes the total number of magic pebbles used in the magic pebbling strategy $\mathcal{P}$.

The following definition, CC^{α}, is an analogous definition to CC as defined by [AS15] (specifically, CC^{α} when $\alpha = 1$ is equivalent to CC) to account for varying costs of memory usage vs. time.

Definition 16 ($\mathbf{CC}^{\alpha}$). *Given a graph, $G \in \mathbb{G}$, and a valid standard/magic pebbling strategy, $\mathcal{P}$, we define the $CC^{\alpha}(G)$ to be*

$$CC^{\alpha}(\mathcal{P}) = (\textsf{p-cc}_{\alpha}(G, \mathcal{P}))\,.$$

Given a graph, $G \in \mathbb{G}$, and a family of valid standard pebbling strategies, $\mathbb{P}$, we define the $CC^{\alpha}(G)$ to be

$$CC^{\alpha}(G) = \min_{\mathcal{P} \in \mathbb{P}} (\textsf{p-cc}_{\alpha}(G, \mathcal{P}))\ ,$$

and, given a family $\mathbb{P}^M$ of valid black-magic pebbling strategies, we define $CC^{\alpha}(G)$ to be

$$CC^{\alpha}(G) = \min_{\mathcal{P}^M \in \mathbb{P}^M} \left(\textsf{p-cc}^M_{\alpha}\left(G, \mathcal{P}^M\right)\right)\ .$$

3 Parallel random oracle model (PROM)

In this paper, we consider two broad categories of computations: *pebbling strategies* and *PROM algorithms*. Specifically, we discussed above the pebbling models and pebble games we use to construct our static memory-hard functions. Now, we define our PROM algorithms.

Prior work has observed the close connections between these two types of computations as applied to DAGs, and our work brings out yet more connections between the two models. In this section, we give an overview of how PROM computations work and define the complexity measures that we apply to PROM algorithms. Some of the complexity measures were introduced by prior work, and others are new in this work.

3.1 Overview of PROM computation

The random oracle model was introduced by [BR93]. When we say random oracle, we always mean a *parallel* random oracle unless otherwise specified.

An *algorithm* in the PROM is a probabilistic algorithm $\mathcal{B}$ which has parallel access to a stateless oracle $\mathcal{O}$: that is, $\mathcal{B}$ may submit many queries in parallel to $\mathcal{O}$. We assume $\mathcal{O}$ is sampled uniformly from an oracle set $\mathbb{O}$ and that $\mathcal{B}$ may depend on $\mathbb{O}$ but not $\mathcal{O}$.

The algorithm proceeds in discrete time-steps called *iterations*, and may be thought to consist of a series of algorithms $(\mathcal{B}_i)_{i \in \mathbb{N}}$, indexed by the iteration i, where each $\mathcal{B}_i$ passes a *state* $\sigma_i \in \{0,1\}^*$ to its successor $\mathcal{B}_{i+1}$. σ_0 is defined to contain the input to the algorithm. We write $|\sigma_i|$ to denote the size, in bits, of σ_i. We write $[\![\sigma_i]\!]$ to denote $\frac{|\sigma_i|}{w}$, where w is the output length of the oracle $\mathcal{O}$. In other words, $[\![\sigma_i]\!]$ is the size of σ_i when counting in words of size w. In each iteration, the algorithm $\mathcal{B}_i$ may make a *batch* $\mathbf{q}_i = (q_{i,1}, \ldots, q_{i,|\mathbf{q}_i|})$ of queries, consisting of $|\mathbf{q}_i|$ individual queries to $\mathcal{O}$, and instantly receive back from the oracle the evaluations of $\mathcal{O}$ on the individual queries, i.e., $(\mathcal{O}(q_{i,1}), \ldots, \mathcal{O}(q_{i,|\mathbf{q}_i|}))$.

At the end of any iteration, $\mathcal{B}$ can append values to a special output register, and it can end the computation by appending a special terminate symbol $\perp$ on

that register. When this happens, the contents y of the output register, excluding the trailing $\perp$, is considered to be the output of the computation. To denote the process of sampling an output, y, provided input x, we write $y \leftarrow \mathcal{B}^{\mathcal{O}}(x)$.

Definition 17 (Oracle functions). *An* oracle function *is a collection* $\mathfrak{f} = \{f^{\mathcal{O}} : D \to R\}_{\mathcal{O}\in\mathbb{O}}$ *of functions with domain D and outputs in R indexed by oracles* $\mathcal{O} \in \mathbb{O}$.

A family of oracle functions *is a set* $\mathcal{F} = \{\mathfrak{f}_\kappa : D_\kappa \to R_\kappa\}_{\kappa\in\mathbb{N}}$ *where each* $\mathfrak{f}_\kappa$ *is indexed by oracles from an oracle set* $\mathbb{O}_\kappa : \{0,1\}^\kappa \to \{0,1\}^\kappa$ *indexed by a security parameter* κ.

Definition 18 (Memory complexity of PROM algorithms). *The* memory complexity of $\mathcal{B}(x;\rho)$ *(i.e., the memory complexity of $\mathcal{B}$ on input x and randomness ρ) is defined as:*

$$\mathsf{mem}_{\mathbb{O}}(\mathcal{B}, x, \rho) = \max_{i\in\mathbb{N}} \{\llbracket \sigma_i \rrbracket\} \ . \tag{1}$$

Definition 19 (Λ-sustained memory complexity of PROM algorithms). *The* Λ-sustained memory complexity of $\mathcal{B}(x;\rho)$ *is defined as:*

$$\mathsf{s\text{-}mem}_{\mathbb{O}}(\Lambda, \mathcal{B}, x, \rho) = |\{i \in \mathbb{N} : |\sigma_i| \geq \Lambda\}| \ . \tag{2}$$

Note that (1) and (2) are distributions over the choice of $\mathcal{O} \leftarrow \mathbb{O}$.

3.2 Functions defined by DAGs

We now describe how to translate a graph construction into a function family, whose evaluation involves a series of oracle calls in the PROM. Any family of DAGs induces a family of *oracle functions* in the PROM, whose complexity is related to the pebbling complexity of the DAG. We first define the syntax of *labeling* of DAG nodes, then define a *graph function family.*

Definition 20 (Labeling). *Let $G = (V, E)$ be a DAG with maximum in-degree δ, let $\mathfrak{L}$ be an arbitrary "label set," and define* $\mathbb{O}(\delta, \mathfrak{L}) = \left(V \times \bigcup_{\delta'=1}^{\delta} \mathfrak{L}^{\delta'} \to \mathfrak{L}\right)$. *For any function $\mathcal{O} \in \mathbb{O}(\delta, \mathfrak{L})$ and any label $\zeta \in \mathfrak{L}$, the $(\mathcal{O}, \zeta)$-labeling of G is a mapping* $\mathsf{label}_{\mathcal{O},\zeta} : V \to \mathfrak{L}$ *defined recursively as follows.*

$$\mathsf{label}_{\mathcal{O},\zeta}(v) = \begin{cases} \mathcal{O}(v, \zeta) & \text{if } \mathsf{indeg}(v) = 0 \\ \mathcal{O}(v, \mathsf{label}_{\mathcal{O},\zeta}(\mathsf{pred}(v))) & \text{if } \mathsf{indeg}(v) > 0 \end{cases} .$$

Definition 21 (Graph function family). *Let $n = n(\kappa)$ and let* $\mathbb{G}_\delta = \{G_{n,\delta} = (V_n, E_n)\}_{\kappa\in\mathbb{N}}$ *be a graph family. We write $\mathbb{O}_{\delta,\kappa}$ to denote the set $\mathbb{O}(\delta, \{0,1\}^\kappa)$ as defined in Definition 20. The* graph function family *of $\mathbb{G}$ is the family of oracle functions* $\mathcal{F}_{\mathbb{G}} = \{\mathfrak{f}_G\}_{\kappa\in\mathbb{N}}$ *where* $\mathfrak{f}_G = \{f_G^{\mathcal{O}} : \{0,1\}^\kappa \to (\{0,1\}^\kappa)^z\}_{\mathcal{O}\in\mathbb{O}_{\delta,\kappa}}$ *and $z = z(\kappa)$ is the number of sink nodes in G. The output of $f_G^{\mathcal{O}}$ on input label $\zeta \in \{0,1\}^\kappa$ is defined to be*

$$f_G^{\mathcal{O}}(\zeta) = \mathsf{label}_{\mathcal{O},\zeta}(\mathsf{sink}(G)) \ ,$$

where $\mathsf{sink}(G)$ *is the set of sink nodes of G.*

3.3 Relating complexity of PROM algorithms and pebbling strategies

Any PROM algorithm $\mathcal{B}$ and input x induce a black-magic pebbling strategy, $\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$)$, called an *ex-post-facto black-magic pebbling strategy.* The way in which this strategy is induced is similar to *ex-post-facto pebbling* as originally defined by [AS15] in the context of the standard pebble game. We adapt their technique for the black-magic game.

Definition 22 (Ex-post-facto black-magic pebbling). *Let $n = n(\kappa)$ and let $\mathbb{G}_\delta = \{G_{n,\delta} = (V_n, E_n)\}_{\kappa\in\mathbb{N}}$ be a graph family. Let $\zeta = \zeta(\kappa) \in \{0,1\}^\kappa$ be an arbitrary input label for the graph function family $\mathcal{F}_\mathbb{G}$. For any $v \in V_n$, define*

$$\mathsf{pre\text{-}lab}_{\mathcal{O},\zeta}(v) = (v, \mathsf{label}_{\mathcal{O},\zeta}(\mathsf{pred}(v))) \ .$$

Let $\mathcal{B}$ be a non-uniform PROM algorithm. Fix an implicit security parameter κ. Let x be an input to $\mathcal{B}$. We now define a magic pebbling strategy induced by any given execution of $\mathcal{B}^\mathcal{O}(x; \$)$, where \$ denotes the random coins of $\mathcal{B}$. Such an execution makes a sequence of batches of random oracle calls (as defined in Section 3.1), which we denote by

$$\mathbf{q}(\mathcal{B}, \mathcal{O}, x, \$) = (\mathbf{q}_1, \ldots, \mathbf{q}_t) \ .$$

The induced black-magic pebbling strategy,

$$\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$) = ((B_0, M_0), \ldots, (B_t, M_t)) \ , \tag{3}$$

is called an ex-post-facto black-magic pebbling*, and is defined by the following procedure.*

1. *$B_0 = M_0 = \varnothing$.*
2. *For $i = 1, \ldots, t$:*
   (a) *$B_i = B_{i-1}$.*
   (b) *$M_i = M_{i-1}$.*
   (c) *For each individual query $q \in \mathbf{q}_i$, if there is some $v \in V_n$ such that $q = \mathsf{pre\text{-}lab}_{\mathcal{O},\zeta}(v)$ and $v \notin P_i$, then "pebble v" by performing the following steps:*
      i. *If $\mathsf{pred}(v) \subseteq M_i \cup B_i$:*
         – *$B_i = B_i \cup \{v\}$.*
      ii. *Else:*
         – *$V = \{v\}$.*
         – *Let V^* be the transitive closure of V under the following operation:*
           $V = V \cup \left(\bigcup_{v'\in V} \mathsf{pred}(v') \cap (M_i \cup B_i)\right)$.
         – *$M_i = M_i \cup V^*$.*
3. *For $i = 1, \ldots, t$:*
   (a) *A node $v \in M_i \cup B_i$ is said to be* necessary at time i *if*

$$\exists j \in [t], q \in \mathbf{q}_j, v' \in V_n \text{ s.t. } \quad j > i \wedge v \in \mathsf{pred}(v') \wedge q = \mathsf{pre\text{-}lab}_{\mathcal{O},\zeta}(v') \wedge \left(\nexists k \in [t], q' \in \mathbf{q}_k \text{ s.t. } i < k < j \wedge q' = \mathsf{pre\text{-}lab}_{\mathcal{O},\zeta}(v)\right) \ .$$

In other words, a node is necessary if its label will be required in a future oracle call, but its label will not be obtained by any oracle query between now and that future oracle call.
Remove from B_i and M_i all nodes that are not necessary at time i.

3.4 Legality and space usage of ex-post-facto black-magic pebbling

The following theorems establish that the space usage of PROM algorithms is closely related to the space usage of the induced pebbling.

We will use the following supporting lemma, also used in prior work such as [AS15,DKW11] (see, e.g., [DKW10] for a proof).

Lemma 1. *Let $B = b_1, \ldots, b_u$ be a sequence of random bits and let $\mathbb{H}$ be a set. Let $\mathcal{P}$ be a randomized procedure that gets a hint $h \in \mathbb{H}$, and can adaptively query any of the bits of B by submitting an index i and receiving b_i as a response. At the end of its execution, $\mathcal{P}$ outputs a subset $S \subseteq \{1, \ldots, u\}$ of $|S| = \varphi$ indices which were not previously queried, along with guesses for the values of the bits $\{b_i : i \in S\}$. Then the probability (over the choice of B and the randomness of $\mathcal{P}$) that there exists some $h \in \mathbb{H}$ such that $\mathcal{P}(h)$ outputs* all *correct guesses is at most $|\mathbb{H}|/2^{\varphi}$.*

Lemma 2 (Legality and magic pebble usage of ex-post-facto black-magic pebbling). *Let $n = n(\kappa)$ and let $\mathbb{G}_\delta = \{G_{n,\delta} = (V_n, E_n)\}_{\kappa\in\mathbb{N}}$ be a graph family. Let $\zeta \in \{0,1\}^\kappa$ be an arbitrary input label for $\mathbb{G}_\delta$. Fix any efficient PROM algorithm $\mathcal{B}$ and input x. With overwhelming probability over the choice of random oracle $\mathcal{O} \leftarrow \mathbb{O}$ and the random coins \$ of $\mathcal{B}$, it holds that the ex-post-facto magic pebbling $\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$)$ consists of valid magic-pebbling moves, and uses fewer than $\chi = \left\lfloor \frac{|x|}{\kappa - \log(q)} + 1 \right\rfloor$ magic pebbles (i.e., for all i, $|M_i| \le \chi$), where q is the number of oracle queries made by $\mathcal{B}(x)$.*

Lemma 3 (Space usage of ex-post-facto black-magic pebbling). *Let $n, \mathbb{G}_\delta, \zeta$ be as in Lemma 2. Fix any PROM algorithm $\mathcal{B}$ and input x. Fix any $i \in [t]$, $\lambda \ge 0$, and define*

$$\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$) = (P_1^{\mathcal{O}}, \ldots, P_t^{\mathcal{O}}) = ((B_1^{\mathcal{O}}, M_1^{\mathcal{O}}), \ldots, (B_t^{\mathcal{O}}, M_t^{\mathcal{O}}))$$

for oracle $\mathcal{O}$. We may omit the superscript $\mathcal{O}$ for notational simplicity. It holds for all large enough κ that the following probability is overwhelming:

$$\Pr\left[\forall i \in [t],\ |P_i| \le \chi'\right] \ ,$$

where $\chi' = \left\lfloor \frac{|\sigma_i|}{\kappa - \log(q)} + 1 \right\rfloor$, q is the number of oracle queries made by $\mathcal{B}$, and the probability is taken over $\mathcal{O} \leftarrow \mathbb{O}$ and the coins of $\mathcal{B}$.

4 Static-memory-hard functions

We now define *static-memory-hard functions.* As mentioned above, prior notions of memory-hardness consider only dynamic memory usage. To model static memory usage, we consider a hash function with two parts $(\mathcal{H}_1, \mathcal{H}_2)$ where $\mathcal{H}_2(x)$ computes the output of the hash function $h(x)$ given oracle access to the output of $\mathcal{H}_1$. This design can be seen to reduce honest party computation time by limiting the hard work to one-off preprocessing phase, while maintaining a large space requirement for password-cracking adversaries. Informally, our guarantee says that unless the adversary stores a specified amount of *static* memory, he must use an equivalent amount of *dynamic* memory to compute h correctly on many outputs. Definition 23 is syntactic and Definition 24 formalizes the memory-hardness guarantee.

Notation PPT stands for "probabilistic polynomial time." For $\boldsymbol{b} \in \{0,1\}^*$, define $\mathsf{Seek}_{\boldsymbol{b}} : \{1, \ldots, |\boldsymbol{b}|\} \to \{0,1\}$ to be an oracle that on input ι returns the ιth bit of $\boldsymbol{b}$.

Definition 23 (Static-memory hash function family (SHF)). *A static-memory hash function family $\mathcal{H}^{\mathcal{O}} = \{h_\kappa^{\mathcal{O}} : \{0,1\}^{w'} \to \{0,1\}^w\}_{\kappa \in \mathbb{N}}$ mapping $w' = w'(\kappa)$ bits to $w = w(\kappa)$ bits is described by a pair of deterministic oracle algorithms $(\mathcal{H}_1, \mathcal{H}_2)$ such that for all $\kappa \in \mathbb{N}$ and $x \in \{0,1\}^n$,*

$$\mathcal{H}_2^{\mathsf{Seek}_{\hat{R}}}(1^\kappa, x) = h_\kappa(x), \ \textit{where } R = \mathcal{H}_1(1^\kappa) \ .$$

(The superscript $\mathcal{O}$ is left implicit.)

The next definition presents a parametrized notion of $(\Lambda, \Delta, \tau, q)$-hardness of an SHF. Before delving into the formal definition, we give a brief intuition of the guarantee provided by Definition 24: any adversary who produces at least q correct input-output pairs of the hash function must *either* have used $\Lambda - \Delta$ static memory *or* incur a requirement of Λ dynamic memory *sustained over* τ time-steps at runtime.

The role of q. The parameter q in Definition 24 serves to capture the intuitive idea that an adversary that uses a certain amount of space could always use that space to directly store output values of h_κ. Clearly, an adversary with an arbitrary input R could very easily output up to $\square|R|\square$ correct output values. Our goal is to lower bound the amount of space needed by an adversary who outputs nontrivially more correct values than that — and q, which is a function of $|R|$, captures how many more.

Definition 24 ($(\Lambda, \Delta, \tau, q)$-hardness of SHF). *Let $\mathcal{H} = \{h_\kappa\}_{\kappa \in \mathbb{N}}$ be a static-memory hash function family described by algorithms $(\mathcal{H}_1, \mathcal{H}_2)$, mapping w' to w bits. $\mathcal{H}$ is* (Λ, Δ, τ)-hard *if for any large enough $\kappa \in \mathbb{N}$, any string $R \in \{0,1\}^{\Lambda - \Delta}$, and any PPT algorithm $\mathcal{A}$, for any set $X = \{x_1, \ldots, x_q\} \subseteq \{0,1\}^{w'}$, there is a negligible ε such that*

$$\Pr_{\mathcal{O},\rho}\Big[\big\{(x_1, h_\kappa(x_1)), \ldots, (x_q, h_\kappa(x_q))\big\} = \mathcal{A}(1^\kappa, R; \rho) \wedge \mathsf{s\text{-}mem}_{\mathbb{O}}(\Lambda, \mathcal{A}, R, \rho) < \tau\Big] < \varepsilon \ .$$

For simplicity, we henceforth assume $w' = w = \kappa$ (i.e., the oracle's input and output sizes are equal to the security parameter) unless otherwise stated.

4.1 Dynamic SHFs

As discussed in detail in the introduction, static memory requirements are orthogonal and complementary to dynamic memory requirements of MHFs as formalized by [AS15]. Given a pebbling-based SHF and a pebbling-based MHF, they can be combined by simple concatenation into a "dynamic SHF," a function that inherits both the static memory requirement of the former and the dynamic memory requirement of the latter, as outlined (informally) next.

Let $\mathcal{H}_{\mathsf{dyn}}^{\mathcal{O}}$ be a dynamic MHF and $(\mathcal{H}_1^{\mathcal{O}}, \mathcal{H}_2^{\mathcal{O}})$ be a SHF family, and the computation of both of these correspond to computing labels of nodes in a DAG as a function of a pebbling algorithm and a random oracle $\mathcal{O}$. We construct a dynamic SHF $\mathcal{H}^{\mathcal{O}}$ that is defined as follows: on input $(1^\kappa, x)$, output $\mathcal{H}_2^{\mathcal{O}(0,\cdot)}(1^\kappa, x)||\mathcal{H}_{\mathsf{dyn}}^{\mathcal{O}(1,\cdot)}(1^\kappa, x)$. The resulting $\mathcal{H}^{\mathcal{O}}$ inherits both the MHF guarantees of $\mathcal{H}_{\mathsf{dyn}}$ and the SHF guarantees of $(\mathcal{H}_1, \mathcal{H}_2)$. Note that importantly, the labels of the nodes in the graphs corresponding to the MHF $\mathcal{H}_{\mathsf{dyn}}^{\mathcal{O}(0,\cdot)}$ and the SHF $(\mathcal{H}_1^{\mathcal{O}(1,\cdot)}, \mathcal{H}_2^{\mathcal{O}(1,\cdot)})$ are independent as the MHF and the SHF use disjoint partitions of the random oracle domain.

Using this method, our SHF constructions can be combined with existing MHF constructions such as [AS15], [ABP17a], [ABP17b], yielding a "best of both worlds" dynamic SHF that enjoys both types of memory-hardness.

5 SHF constructions

A first attempt What if we pebble a hard-to-pebble graph, and then let $R_{k,i} = H(P(k), i)$ where $P(k)$ is the entire pebbling of the graph (on input k and iteration i is the i-th call to the hash function H)? This would in fact work in the random oracle model where the random oracle takes arbitrary-length input. However, in practice, hash functions do not take arbitrary-length input. While constructions like Merkle-Damgård [Mer79] and sponge [BDPA08] can transform a fixed-input-length hash function into one that takes arbitrary-length inputs, the resulting function does *not* behave like a random oracle even if the fixed-length hash function does.Moreover, the computation graphs of known length-expanding transformations such as Merkle-Damgård and sponge functions require very little space to compute. For instance, the computation graph of the Merkle-Damgård construction is a binary tree and the computation graph of the sponge function is a caterpillar graph both of which take logarithmic and constant space, respectively, to compute. Thus, we have to use special constructions to achieve the local-hardness properties we need.

Recall from Definition 13 that the property we want is this "locally hard to access" notion, meaning that if an adversarial party chooses to not store the static part of our hash function which they obtain from performing the "preprocessing" computation associated with $\mathcal{H}_1$, then they must use the same memory and

sustained time to recompute the function when our static-memory-hard function is called on *any subset of inputs* larger than the memory used to store the preprocessed computation. We achieve this desired property in our $\mathcal{H}_1$ functions using two novel DAG constructions, one of which is optimal for a specific graph class and the other we conjecture to be optimal for all general graph classes.

5.1 $\mathcal{H}_1$ constructions

We first note the differences between the graph constructions we present here and the constructions presented in previous literature [AS15,ACK+16,ABP17a,DFKP15]. Firstly, many of the constructions presented in previous work feature a single target node. This is reasonable in the context of memory-hard functions since both the honest party and the adversary must compute the hash function dynamically (obtaining a single label as the output of the function) on each input. However, in our context of static-memory-hard functions, single-target-node constructions do not make sense. Secondly, our constructions differ from even the multiple target node constructions presented in the literature (specifically, the constructions of [DFKP15]) since prior constructions mainly focused on finding graphs that have large memory vs. time tradeoffs.

Our constructions are designed with the goal that any adversary that does not store almost all the target labels must dynamically use *the same amount of space as needed to store all the labels* to compute the hash function (while *still incurring a cost in runtime*). Moreover, our constructions based on local hardness ensure a stronger guarantee than the constructions in [DFKP15]; in our case, one must use at least S space (for some definition of S) to compute *any* given subset of targets larger than one's current memory usage, whereas in their case, they use S space to compute some subset of targets chosen uniformly at random. Therefore, our specifications are stronger in that we provide a space bound as well as a time bound for adversaries; and moreover, for *honest* parties, the time cost is only a one-time setup cost. We prove our pebbling costs in terms of the black-magic pebble game (defined in Section 2) as opposed to the standard pebble game used in previous works. Most notably, this means that in all of our constructions, the pebbling number is upper bounded by the number of targets (since one can always just pebble the targets with magic pebbles).

We begin with some simple and clean constructions of $\mathcal{H}_1$ based on pebbling constructions that exist in the literature. We first prove a lemma regarding the minimum number of pebbles used in the PROM model and the minimum number of pebbles used in the sequential memory model. This is useful in more than one way: (1) it tells us that parallelization does not save the adversary in space so honest parties (who can only compute a constant number of labels at a time) and adversaries (who can compute an arbitrary number of labels at the same time) operate under the same space constraints and (2) it allows us to directly compare sustained time complexities between adversaries and honest parties with respect to space usage .

Lemma 4 (Standard Pebbling Sequential/Parallel Equivalence). *Given a DAG* $G = (V, E)$*,* $\mathbf{P}_{\mathrm{s}}(G, T) = \mathbf{P}^{\|}_{\mathrm{s}}(G, T)$ *where* $\mathbf{P}_{\mathrm{s}}(G, T)$ *is defined to be the*

minimum standard pebbling space complexity in the sequential model, and we define $\mathbf{P}_{\mathrm{s}}^{\parallel}(G, T)$ *to be the minimum standard pebbling space complexity in the parallel model.*

We use Lemma 4 to prove an equivalent lemma for the black-magic pebble game below.

Lemma 5 (Black-Magic Pebbling Sequential/Parallel Equivalence). *Given a DAG* $G = (V, E)$*,* $\mathbf{P}_{\mathrm{s}}(G, |T|, T) = \mathbf{P}_{\mathrm{s}}{}^{\parallel}(G, |T|, T)$ *where* $\mathbf{P}_{\mathrm{s}}(G, |T|, T)$ *was defined to be the minimum black-magic pebbling space complexity in the sequential model, and we define* $\mathbf{P}_{\mathrm{s}}{}^{\parallel}(G, |T|, T)$ *to be the minimum black-magic pebbling space complexity in the parallel model.*

Now, we jump into our constructions. We first provide a simple construction and show why this construction is not optimal. In addition, we define some subgraph components in the pebbling literature that are important subcomponents of our constructions.

A failed attempt at $\mathcal{H}_1$ We first provide a failed attempt at constructing $\mathcal{H}_1$ due to the large amount of time that is needed to compute the function (for the sequential honest party) with respect to the amount of memory needed to store the output of the function. In other words, this construction is problematic in the sense that an exponential number of steps is necessary to compute the stored results of the function from scratch for the honest party but the adversary with parallel processing time can compute the function from scratch in linear time. Although the honest party could obtain the results of the preprocessing (i.e. the static part of the hash function) from elsewhere, we must ensure that they can still feasibly compute $\mathcal{H}_1$ themselves in the event that they do not trust any of the sources from which they can obtain the static data.

Intuitively, our failed attempt at constructing $\mathcal{H}_1$ is a series of binary search trees. From here onwards, we describe all constructions of $\mathcal{H}_1$ as a directed acyclic graph with n nodes and later use our theorems above to prove static memory hardness from our constructed DAGs.

Graph Construction 1 (Composite Binary Tree DAG)**.** *Let* B_h^C *be a composite binary tree DAG with height* h *constructed in the following way where* T *is the number of targets of our DAG. Let* $s = |T|$*. In our intended construction* $h = s$*.*

1. *Let the set of nodes be* V*. Let the set of edges be* E*.*
2. *Create* $(s + 1)2^{h-1} + s$ *nodes.*
3. *Create* $s + 1$ *binary search trees using* $(s + 1)2^{h-1}$ *nodes in total where edges are directed from children to parents in each binary tree. Let* r_i *for* $i \in [1, s+1]$ *be the roots of these binary search trees.*
4. *Order the remaining nodes in some arbitrary order, let* s_j *be the* j*th node in this order for* $j \in [1, s]$*.*
5. *Create directed edges* (r_i, s_i) *and* $(r_{i+1 \bmod s}, s_i)$ *for all* $i \in [1, s]$*.*

Given any binary search tree with height h, the minimum number of pebbles necessary to pebble the tree is h (assuming a 'tree' with one node has height 1)

using the rules of the standard pebble game. Therefore, to ensure that the apex of the tree is pebbled and that both the honest party and the adversary both use h space to pebble the apex, the number of leaves necessary at the base of the tree is 2^{h-1}. If we suppose that the computationally weak honest party (who does not build special circuits) can only evaluate a constant number of random oracle calls at a time (place a constant number of pebbles), the number of sequential evaluations necessary for the honest party is $\geq \Omega(2^h)$ which is infeasible to accomplish. In constrast, the adversary only has to make $O(h)$ parallel random oracle calls, an exponential factor difference between the honest party and the adversary! Such a construction fails since it is clearly infeasible for the honest party since they would never be able to compute all target values of $\mathcal{H}_1$ from scratch (since this computation requires exponential time for the honest party). Thus, we would like a construction that has the same minimum space requirement but also small sequential evaluation time. We prove a better (but also simply defined) construction below.

Cylinder construction We make use of what is defined in the pebbling literature as a *pyramid graph* [GLT80] in constructing our *cylinder graph*. The key characteristic of the pyramid graph we use is that the number of pebbles that is required to pebble the apex of the pyramid is equal to the height of the pyramid [GLT80] using the rules of the standard pebble game. Note that a pyramid by itself is not useful for our purposes since the black-magic pebbling space complexity of a pyramid with one apex is 1. Therefore, we need to be able to use the pyramid in a different construction that uses superconstant number of pebbles in the magic pebble game in order to successfully pebble all target nodes.

Graph Construction 2 (Illustrated in Fig. 2)**.** *Let Π_h^C be a* cylinder *graph with height h. We define Π_h^C as follows:*

1. *Create $2h^2$ nodes. Let this set of $2h^2$ nodes be V.*
2. *Arrange the nodes in V into $2h$ levels of h nodes each, ranging from level 0 to level $2h-1$. Let the j-th node in level i be v_i^j. Create directed edges $(v_i^{j \bmod h}, v_{i+1}^{j \bmod h})$ and $(v_i^{j \bmod h}, v_{i+1}^{(j+1) \bmod h})$ for all $i \in [0, 2h-2]$. Let this set of edges be E.*

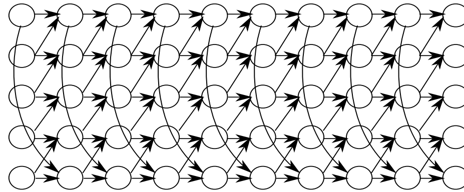


Fig. 2: Cylinder construction (Def. 2) for $h = 5$.

Lemma 6. *Given a cylinder graph with height h, Π_h^C, $\mathbf{P}_{\mathrm{s}}(\Pi_h^C, T) \geq h$.*

Lemma 7. $\mathbf{P}_{\text{opt-ss}}(\Pi_h^C, T) \geq 2h$.

Theorem 3. *Using the rules of the standard pebble game, h pebbles are necessary for at least h parallel steps to pebble* any *target of a height $2h$ cylinder graph, Π_h^C.*

Theorem 4. $\mathbf{P}_{\mathrm{s}}(\Pi_h^C, |T|, T) \geq h$ *where Π_h^C is defined as in Def. 2 where $|S| = |T| = h$.*

As a simple extension of our theorem and proof above, we get Corollary 1. Moreover, as an extension of the proof given for Theorem 4 that all magic pebbles are placed on targets and from Theorem 3, we obtain Corollary 2.

Corollary 1. *Given a cylinder $G = (V, E)$ as constructed in Graph Construction 2, G is incrementally hard: $\mathbf{P}_{\mathrm{s}}(G, |C| - 1, C) \geq |T|$ for any subset $C \subseteq T$.*

Corollary 2. *Given a cylinder $G = (V, E)$ as constructed in Graph Construction 2, $\mathbf{P}_{\text{opt-ss}}(G, |C| - 1, C) = \Theta(|T|)$ for all subsets of $C \subseteq T$.*

A logical question to ask after constructing our very simple hash function based on a cylinder graph is whether such a construction is optimal in terms of graph-optimal sustained complexity *and* follows our requirements for a static-memory-hard hash function. As it turns out, the graph-optimal sustained complexity of a cylinder graph is optimal in the class of layered graphs. In other words, if we choose to use layered graphs in our constructions, then we cannot hope to get a better memory and time guarantee. From an implementation and practical standpoint, layered graphs are easier to implement and hence this result has potential practical applications (as more complicated constructions need to consider memory allocation factors in the real-life implementation, not considered in the theoretical model).

Theorem 5. *Given a* layered graph*, $G = (V, E)$, if the number of target nodes is $|T| = s$ and $\mathbf{P}_{\mathrm{s}}(G, s, T) \geq s$, then $|V| = \Omega(s^2)$. A* layered graph *is one such that the vertices can be partitioned into layers and edges only go between vertices in consecutive layers.*

Thus, our construction of the cylinder graph is optimal in terms of amount of memory used in the asymptotic sense for the class of layered graphs. An open question is whether this is also optimal when we consider the larger class of all DAGs.

Open Question. *Does Thm 5 also hold for general graphs with bounded in-degree 2?*

Given the impossibility of providing a better space guarantee for layered graphs, we provide a general (non-layered) construction that transforms a graph from a certain class into another graph with the same space guarantee as in Theorem 5. Furthermore, we provide an example below that has the same space guarantees but a better time guarantee.

Layering *shortcut-free* graphs We now show how to convert any *shortcut-free* DAG, $G = (V, E)$, with $\mathbf{P}_{\mathrm{s}}(G, T) = s$ and one target node (i.e. $|T| = 1$) into a DAG, $G' = (V', E')$, with $|T'| = s$ targets and $\mathbf{P}_{\mathrm{s}}(G', s, |T'|) = s$.

Definition 25 (Shortcut-Free Graphs). *Let $G = (V, E)$ be a DAG where $\mathbf{P}_{\mathrm{s}}(G, T) \geq s$. Let $t_s^{\mathcal{P}}$ be the last time step that exactly s pebbles must be on G during any normal and regular pebbling strategy, $\mathcal{P}$, (see our full version [DLP18] and [GLT80,DL17]) that uses s pebbles. More specifically, let Let X be the union of the set of nodes that are pebbled at $t_s^{\mathcal{P}}$ for all normal and regular strategies $\mathcal{P}$: $X = \bigcup_{\mathcal{P} \in \mathbb{P}} P_{t_s^{\mathcal{P}}}$. Let D be the set of descendants of nodes of X. A DAG is* shortcut-free *if $|X| \leq s$ and given $s_1 < s$ pebbles placed on any subset $X_1 \subset X$, no normal and regular strategy uses less than $s - s_1$ pebbles to pebble $D \cup (X \backslash X_1)$.*

Graph Construction 6. *Given a shortcut-free DAG, $G = (V, E)$, with $\mathbf{P}_{\mathrm{s}}(G, T) = s$ and $|T| = 1$, we create a DAG, $G' = (V', E')$, with the following vertices and edges and with the set of targets T' where $|T'| = s$. Let X be defined as in Definition 25.*

1. *V' is composed of the nodes in V and $s - 1$ copies of $X \cup D$. Let the i-th copy of X be X_i (the original is X_0) and let the i-th copy of $x \in X_i$ be x_i.*
2. *E' is composed of the edges in E and the following directed edges. If $(v, w) \in E$ and $v, w \in X$, then create edges $(v_i, w_i) \in E'$ for all $i \in [1, s - 1]$. Create edges $(u, v_i) \in E'$ if $(u, v) \in E$ and $u \in V \backslash X, D$.*
3. *The set of targets T' is the union of the set of targets of the different copies: $T' = \bigcup_{i=0}^{s-1} T_i$.*

Using the above construction, we have created a graph $G' = (V', E')$ where $|V'| = |V| + (s - 1)(|D| + |X|)$ and $|T'| = s$.

Theorem 7. *Given a shortcut-free DAG $G = (V, E)$ with $\mathbf{P}_{\mathrm{s}}(G, T) = s$ and $|T| = 1$, the construction produced by Graph Construction 6 produces a DAG $G' = (V', E')$ such that $\mathbf{P}_{\mathrm{s}}(G', s, |T|) = s$.*

If $D = \Theta(s)$ and $s = O(\sqrt{|V|})$, then $|V'| = \Theta(s^2 + |V|)$ which has a better sustained time guarantee than our cylinder construction.

We first note that the sustained memory graphs presented in [ABP17a] *do not* achieve optimal local memory hardness because $X \cup D$ (as defined in Definition 6) is $\Theta(n)$ (since the sources are the ones that remain pebbled in their construction). Thus, we would like to provide a construction of a shortcut-free DAG where $|X \cup D| = \Theta(s)$. Note that the size of $X \cup D$ will always be $\Omega(s)$, trivially. We now provide a definition of a shortcut-free graph class G that can be transformed using Definition 6.

Graph Construction 8 (Illustrated in Fig. 3)**.** *Let $G = (V, E)$ be a graph defined by parameter s and in-degree 2 with the following set of vertices and edges:*

1. *Create a height s pyramid. Let r_i be the root of a* subpyramid *(i.e. a pyramid that lies in the original height s pyramid) with height $i \in [2, s]$. One can pick any set of these subpyramids.*

2. *Topologically sort the vertices in each level and create a path through the vertices in each level (see Fig. 3). Replace any in-degree-3 nodes with a pyramid of height 3, with a 6-factor increase in the number of vertices.*
3. *Create $c_1 s$ additional nodes for some constant $c_1 \geq 2$ (in Fig. 3, $c_1 = 6$). Label these nodes v_j for all $j \in [1, c_1 s]$.*
4. *Create directed edges (r_s, v_1) and $(r_i, v_{k(i-1)})$ for all $k \in [1, s]$.*
5. *Create $s - 1$ additional nodes. Let these nodes be w_l for all $l \in [1, s-1]$.*
6. *Create directed edges $(v_{c_1 s}, w_1)$ and (r_i, w_{i-1}) for all $i \in [2, s]$.*
7. *The target node is w_{s-1}.*

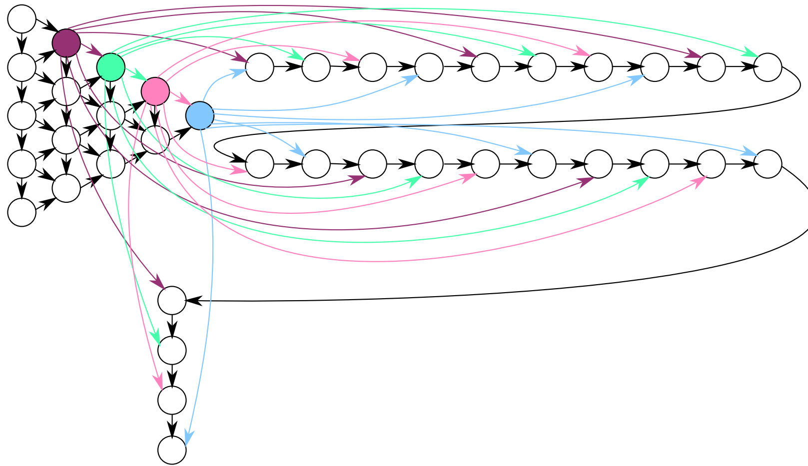


Fig. 3: Example of a time optimal graph family construction as defined in Def. 8. Here, $s = 5$.

Lemma 8. *Given a DAG $G = (V, E)$ and a parameter s where G is defined by Definition 8, $\mathbf{P}_{\mathrm{s}}(G, T) = s$.*

Before we prove that $G = (V, E)$ created by Definition 8 with parameter s is shortcut-free, we first prove the following stronger lemma which will help us prove that G is shortcut-free.

Lemma 9. *Let $G = (V, E)$ be a graph created using Definition 8 with parameter s. Given a normal strategy $\mathcal{P}$ to pebble G, when v_q for $q \in [1, c_1 s]$ is pebbled at some time step, black pebbles are present on all nodes in $[r_i, r_s]$ where $i = (q \bmod s-1) + 1$ from the time when v_1 is pebbled to when v_q is pebbled.*

Lemma 10. *Given a DAG $G = (V, E)$ and a parameter s where G is defined by Definition 8, G is shortcut-free.*

Theorem 9. *s pebbles are necessary for at least $\Theta(s^2)$ parallel steps to pebble any target of G'.*

We create $G' = (V', E')$ from G (as constructed using Definition 8) using Definition 6 , resulting in a graph with $\Theta(s^2)$ total nodes.

Theorem 10. $\mathbf{P}_{\mathrm{s}}(G', s, T) = s$.

By the proof that G' is shortcut-free, we obtain the following corollary that G' is also incrementally hard. Moreover, Corollary 4 follows directly from the proof of Theorem 7.

Corollary 3. *Given a graph $G = (V, E)$ as constructed in Graph Construction 8, G is incrementally hard: $\mathbf{P}_{\mathrm{s}}(G, |C| - 1, C) \geq |T|$ for any subset $C \subseteq T$.*

The following corollary about the graph-optimal sustained time complexity is proven directly from the proof of Lemma 9 and Theorem 9 that if less than $\frac{s}{2}$ magic pebbles are on the pyramid, then half the pyramid must be rebuilt resulting in $\Theta(s^2)$ time-steps in which s pebbles are on the graph; thus proving for the cases when $|C| - 1 < \frac{s}{2}$. We now prove the case when $|C| - 1 \geq \frac{s}{2}$.

Corollary 4. *Given a graph $G = (V, E)$ as constructed in Graph Construction 8, $\mathbf{P}_{\text{opt-ss}}(G, |C| - 1, C) = \Theta(|V|)$ for all subsets of $C \subseteq T$.*

5.2 $\mathcal{H}_2$ construction

Our construction of $\mathcal{H}_2$ is presented in Algorithm 1.

Algorithm 1 $\mathcal{H}_2$

On input $(1^\kappa, x)$ and given oracle access to Seek_R (where R is the string outputted by $\mathcal{H}_1$):

1. Let $[\![R]\!] = |R|/w$ be the length of R in words.
2. Query the random oracle to obtain $\rho_0 = \mathcal{O}(x)$ and $\rho_1 = \mathcal{O}(x + 1)$.
3. Use ρ_0 to sample a random $\iota \in [[\![R]\!]]$.
4. Query the Seek_R oracle to obtain $y' = \mathsf{Seek}_R(\iota)$.
5. Output $y' \oplus \rho_1$.

Lemma 11. *For any R, the output distribution of $\mathcal{H}_2$ is uniform over the choice of random oracle $\mathcal{O} \leftarrow \mathbb{O}$.*

Remark 6. Lemma 11 is important as an indication that our SHF construction "behaves like a random oracle." The memory-hardness guarantee alone does not assure that the hash function is suitable for cryptographic hashing: e.g., a modified version of $\mathcal{H}_2$ which directly outputted y' instead of $y' \oplus \rho_1$ would still satisfy memory-hardness, but would be an awful hash function (with polynomial size codomain). The inadequacy of existing memory-hardness definitions for assuring that a function "behaves like a hash function" is discussed by [AT17].

5.3 Proofs of hardness of SHF Constructions

We now prove the hardness of our graph constructions given earlier in Section 5.

We begin by stating two supporting lemmata. The first is due to Erdős and Rényi [ER61], on the topic of the Coupon Collector's Problem.

Lemma 12 ([ER61]). *Let Z_n be a random variable denoting the number of samples required, when drawing uniformly from a set of n distinct objects with replacement, to draw each object at least once. Then for any c,* $\lim_{n\to\infty} \Pr[Z_n < n \log n + cn] = e^{-e^{-c}}$.

Corollary 5. *Let $Z_{n,k}$ be a random variable denoting the number of samples required, when drawing uniformly from a set of n distinct objects with replacement, to have drawn at least $k \in [n]$ distinct objects. Let $q \in \omega(k \log k)$. Then $\Pr[Z_{n,k} < q]$ is overwhelming (in k).*

Theorems 11–14 state the static-memory-hardness of our SHF constructions based on Graph Constructions 2 and 8.

Theorem 11. *Define a static-memory hash function family $(\mathcal{H}_1, \mathcal{H}_2)$ as follows: let $\mathcal{H}_1$ be the graph function family $\mathcal{F}_{\Pi_h^C}$ (Graph Construction 2), and let $\mathcal{H}_2$ be as defined in Algorithm 1. Let $\mathcal{H} = \{h_\kappa\}_{\kappa\in\mathbb{N}}$ be the static-memory hash function family described by $(\mathcal{H}_1, \mathcal{H}_2)$. Let $\hat{\kappa} = \kappa - \xi \log(\kappa)$ for any $\xi \in \omega(1)$, let $\hat{\Lambda}, \tau \in \Theta(\sqrt{n})$, and let $q \in \omega(\Lambda \log \Lambda)$. Then $(\mathcal{H}_1, \mathcal{H}_2)$ is $(\hat{\kappa}\hat{\Lambda}, \hat{\kappa}, \tau, q)$-hard.*

Theorem 12. *Define a static-memory hash function family $(\mathcal{H}_1, \mathcal{H}_2)$ as follows: let $\mathcal{H}_1$ be the graph function family $\mathcal{F}_G$ (Graph Construction 8), and let $\mathcal{H}_2$ be as defined in Algorithm 1. Let $\hat{\kappa} = \kappa - \xi \log(\kappa)$ for any $\xi \in \omega(1)$, let $\hat{\Lambda} \in \Theta(\sqrt{n})$, let $\tau \in \Theta(n)$, and let $q \in \omega(\Lambda \log \Lambda)$. Then $(\mathcal{H}_1, \mathcal{H}_2)$ is $(\hat{\kappa}\hat{\Lambda}, \hat{\kappa}, \tau, q)$-hard.*

The parameter q is suboptimal in Theorems 11 and 12. We can achieve optimality (i.e., $q = [\![|R|]\!]$) by the following alternative construction of $\mathcal{H}_2$: make $q' = \omega(\log(\kappa))$ random calls instead of just one call to the Seek oracle in Step 4. To preserve the output size of h_κ, it may be useful to reduce the size of node labels by a corresponding factor of q'. This can be achieved by truncating the random oracle outputs used to compute labels in Definition 20. The description of this altered $\mathcal{H}_2^{q'}$ and the definition of graph function family $\mathcal{F}q'_G$ with shorter labels are given in our full version [DLP18].

Theorem 13. *Define a static-memory hash function family $(\mathcal{H}_1, \mathcal{H}_2)$ as follows: let $\mathcal{H}_1$ be the graph function family $\mathcal{F}^{\kappa/q'}_{\Pi_h^C}$ (Graph Construction 2), and let $\mathcal{H}_2$ be $\mathcal{H}_2^{q'}$ as defined in our full version [DLP18] for some $q' \in \omega(\log \Lambda)$. Let $\hat{\kappa} = \kappa - \xi \log(\kappa)$ for any $\xi \in \omega(1)$, let $\hat{\Lambda}, \tau \in \Theta(\sqrt{n})$, and let $q = \Lambda$. Then $(\mathcal{H}_1, \mathcal{H}_2)$ is $(\hat{\kappa}\hat{\Lambda}, \hat{\kappa}, \tau, q)$-hard.*

Theorem 14. *Define a static-memory hash function family $(\mathcal{H}_1, \mathcal{H}_2)$ as follows: let $\mathcal{H}_1$ be the graph function family $\mathcal{F}^{\kappa/q'}_G$ (Graph Construction 8), and let $\mathcal{H}_2$ be $\mathcal{H}_2^{q'}$ as defined in our full version [DLP18] for some $q' \in \omega(\log \Lambda)$. Let $\hat{\kappa} = \kappa - \xi \log(\kappa)$ for any $\xi \in \omega(1)$, let $\hat{\Lambda} \in \Theta(\sqrt{n})$, let $\tau \in \Theta(n)$, and let $q = \Lambda$. Then $(\mathcal{H}_1, \mathcal{H}_2)$ is $(\hat{\kappa}\hat{\Lambda}, \hat{\kappa}, \tau, q)$-hard.*

Acknowledgements

We are grateful to Jeremiah Blocki, Krzysztof Pietrzak, and Joël Alwen for valuable feedback on earlier versions of this paper. We thank Ling Ren for helpful technical discussions. We also thank Erik D. Demaine and Shafi Goldwasser for their advice on this paper. Finally, we thank our anonymous reviewers for insightful comments.

Sunoo's research is supported by NSF MACS (CNS-1413920), DARPA IBM (W911NF-15-C-0236), SIMONS Investigator Award Agreement Dated June 5th, 2012, and the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370.

References

AAC+17. Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. 10625:357–379, 2017.

AB16. Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 241–271. Springer, 2016.

ABH17. Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical graphs for optimal side-channel resistant memory-hard functions. In *CCS*, pages 1001–1017. ACM, 2017.

ABP17a. Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In *EUROCRYPT (3)*, volume 10212 of *Lecture Notes in Computer Science*, pages 3–32, 2017.

ABP17b. Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. *CoRR*, abs/1705.05313, 2017.

ACK+16. Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. On the complexity of scrypt and proofs of space in the parallel random oracle model. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 358–387. Springer, 2016.

ACP+16. Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. *IACR Cryptology ePrint Archive*, 2016:989, 2016.

ADN+10. Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 113–134. Springer, 2010.

AdRNV17. Joël Alwen, Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. Cumulative space in black-white pebbling and resolution. In *ITCS*, volume 67 of *LIPIcs*, pages 38:1–38:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

ADW09. Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Survey: Leakage resilience and the bounded retrieval model. In Kaoru Kurosawa, editor, *Information Theoretic Security, 4th International Conference, ICITS 2009, Shizuoka, Japan, December 3-6, 2009. Revised Selected Papers*, volume 5973 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

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

AT17. Joël Alwen and Björn Tackmann. Moderately hard functions: Definition, instantiations, and applications. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 493–526. Springer, 2017.

BDPA08. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 181–197, 2008.

Ben89. Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

BK15. Alex Biryukov and Dmitry Khovratovich. Tradeoff cryptanalysis of memory-hard functions. In *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 633–657. Springer, 2015.

BKR16. Mihir Bellare, Daniel Kane, and Phillip Rogaway. Big-key symmetric encryption: Resisting key exfiltration. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 373–402. Springer, 2016.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73. ACM, 1993.

BZ16. Jeremiah Blocki and Samson Zhou. On the computational complexity of minimal cumulative cost graph pebbling. *CoRR*, abs/1609.04449, 2016.

BZ17. Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of argon2i. In *TCC (1)*, volume 10677 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2017.

CDD+07. David Cash, Yan Zong Ding, Yevgeniy Dodis, Wenke Lee, Richard J. Lipton, and Shabsi Walfish. Intrusion-resilient key exchange in the bounded retrieval model. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 479–498. Springer, 2007.

Cha73. Ashok K. Chandra. Efficient compilation of linear recursive programs. In *SWAT (FOCS)*, pages 16–25. IEEE Computer Society, 1973.

CLW06. Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In Halevi and Rabin [HR06], pages 225–244.

Coo73. Stephen A. Cook. An observation on time-storage trade off. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 29–33, New York, NY, USA, 1973. ACM.

CS74. Stephen Cook and Ravi Sethi. Storage requirements for deterministic / polynomial time recognizable languages. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 33–39, New York, NY, USA, 1974. ACM.

DFKP15. Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. *Proofs of Space*, pages 585–605. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

DKW10. Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable and uncomputable functions. *IACR Cryptology ePrint Archive*, 2010:541, 2010.

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

DL17. Erik D. Demaine and Quanquan C. Liu. Inapproximability of the standard pebble game and hard to pebble graphs. In *Proceedings of the 16th International Symposium on Algorithms and Data Structures, WADS '17*, volume 10389 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2017.

DLP18. Thaddeus Dryja, Quanquan C. Liu, and Sunoo Park. Static-memory-hard functions and nonlinear space-time tradeoffs via pebbling. *IACR Cryptology ePrint Archive*, 2018:205, 2018.

Dzi06. Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In Halevi and Rabin [HR06], pages 207–224.

ER61. Paul Erdős and Alfréd Rényi. On a classical problem of probability theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 6:215–220, 1961.

FLW13. Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena: A memory-consuming password scrambler. *IACR Cryptology ePrint Archive*, 2013:525, 2013.

GLT80. John R. Gilbert, Thomas Lengauer, and Robert Endre Tarjan. The pebbling problem is complete in polynomial space. volume 9, pages 513–524, 1980.

HPV77. John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *J. ACM*, 24(2):332–337, April 1977.

HR06. Shai Halevi and Tal Rabin, editors. *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*. Springer, 2006.

JWK81. Hong Jia-Wei and H. T. Kung. I/o complexity: The red-blue pebble game. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, STOC '81, pages 326–333, 1981.

LT82. Thomas Lengauer and Robert E. Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *J. ACM*, 29(4):1087–1130, October 1982.

Mer79. Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford, CA, USA, 1979. AAI8001972.

Nor12. Jakob Nordström. On the relative strength of pebbling and resolution. *ACM Trans. Comput. Log.*, 13(2):16:1–16:43, 2012.

Nor15. Jakob Nordstrom. New wine into old wineskins: A survey of some pebbling classics with supplemental results. 2015.

Per09. Colin Percival. Stronger key derivation via sequential memory-hard functions, 2009. Presented at BSDCan 2009. Available online at: `http://www.tarsnap.com/scrypt/scrypt.pdf`.

PH70. Michael S. Paterson and Carl E. Hewitt. Record of the project mac conference on concurrent systems and parallel computation. chapter Comparative Schematology, pages 119–127. ACM, New York, NY, USA, 1970.

Pip82. Nicholas Pippenger. Advances in pebbling (preliminary version). In *ICALP*, volume 140 of *Lecture Notes in Computer Science*, pages 407–417. Springer, 1982.

Pot17. Aaron Potechin. Bounds on monotone switching networks for directed connectivity. *J. ACM*, 64(4):29:1–29:48, 2017.

RD17. Ling Ren and Srinivas Devadas. Bandwidth hard functions for ASIC resistance. In *TCC (1)*, volume 10677 of *Lecture Notes in Computer Science*, pages 466–492. Springer, 2017.

Set75. Ravi Sethi. Complete register allocation problems. *SIAM J. Comput.*, 4(3):226–248, 1975.

SS79a. John E. Savage and Sowmitri Swamy. Space-time tradeoffs for oblivious integer multiplication. In Hermann A. Maurer, editor, *Automata, Languages and Programming*, pages 498–504, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

SS79b. Sowmitri Swamy and John E. Savage. Space-time tradeoffs for linear recursion. In *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '79, pages 135–142, New York, NY, USA, 1979. ACM.

Tom81. Martin Tompa. Corrigendum: Time-space tradeoffs for computing functions, using connectivity properties of their circuits. *J. Comput. Syst. Sci.*, 23(1):106, 1981.

Val77. Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings*, pages 162–176, 1977.