Memory hard functions and persistent memory hardness

Niels Kornerup

Spring 2022

Abstract

Moderately hard functions are a common cryptographic tool for tasks such as password hashing, spam prevention, and proofs of work. The goal of a moderately hard function is that it should be relatively easy for an honest party to compute a small number of times, but prohibitively expensive for an adversary to brute force. Memory hard functions (MHFs) are a type of moderately hard function that rely on using large amounts of space. Such functions become prohibitive for an adversary to compute in parallel, making them ideal candidates for tasks like password hashing. A well designed memory hard function should not only require a large amount of space, but require it in a large number of steps. This idea has been formalized in the cumulative and consistent space complexities. In this survey we look at two MHF constructions and their upper and lower bounds under different metrics of space and time complexity.

1 Introduction

Space complexity is often a factor that is overlooked when investigating the efficiency of a computation. Often times we are able to prove space-time trade-offs, allowing tasks to be computed in a fewer number of steps at a cost of additional space [Hel80, CDG18]. There are lower bounds for the space time complexity of important computational tasks such as sorting [BST98]. Additionally we have many problems where under cryptographic assumptions we are able to lower bound the amount of space (or a steep space time trade-off) needed to solve the problem, creating what is known as a memory hard functions [ABP17, BCGS16].

Sometimes when discussing the space complexity of a problem we are only concerned with the maximum space usage. Such a metric can be misleading about the complexity of a problem, as an adversary can abuse computation that requires only a single spike in memory usage [AS14]. Rather than looking at the maximum space usage of an algorithm, it can be informative to look at the space used over time. Some metrics for this include the cumulative complexity (CC), which is the sum of the space used per step, and counting the number of high memory steps in the computation [AS14, RD16]. In this survey I investigate the assumptions underpinning MHFs, existing constructions, and attacks.

2 Preliminaries

2.1 Space time complexity

There are a couple of measures for the space complexity of algorithms. These definitions can be easily generalized to average or worst case inputs, as well as to lower bounds over all algorithms for some problem instance.

Definition 1. *Let $\mathcal{A}$ be an algorithm that uses $S(\mathcal{A}_t(x))$ space in the t'th step for input x. The* transient space *of $\mathcal{A}$ on input x is $S(\mathcal{A}(x)) = \max_t S(\mathcal{A}_t(x))$.*

Often times it is possible to solve a problem with less space by using a larger number of steps. This notion is captured in a space time trade-off. While there are many possible trade-offs between space and time for algorithms, we will focus on the duration of space usage for a computation. This is captured in the space time complexity.

Definition 2. *Let $\mathcal{A}$ be an algorithm that runs in $T(\mathcal{A}(x))$ steps. The* space time *of $\mathcal{A}$ on input x is $ST(\mathcal{A}(x)) = S(\mathcal{A}(x))T(\mathcal{A}(x))$.*

The above two definitions are standard measures of complexity in algorithms. When analyzing memory hardness, we often want a stronger notion than the above definitions can provide. For example, it is possible to amortize spikes in memory cost by staggering multiple instances of an algorithm on independent inputs [AS14]. We now give our definition of cumulative complexity, which was first formally defined in [AS14].

Definition 3 (cumulative complexity)**.** *The* cumulative complexity *of $\mathcal{A}$ on input x is $cc(\mathcal{A}(x)) = \sum_t S(\mathcal{A}_t(x))$.*

Cumulative complexity has some nice properties. For example, the cumulative complexity for solving n independent instances of algorithm $\mathcal{A}$ is given by $ncc(\mathcal{A})$. While cumulative complexity protects against parallel instance attacks, the definition tells us very little about possible space-time trade-offs. Consistent memory hardness gives us a way to guarantee that a large amount of memory is used in many steps [RD16].

Definition 4. *The S'* consistent memory *of $\mathcal{A}$ on input x is the number of steps t where $S(\mathcal{A}_t(x)) \geq S'$. We denote this value as $S_{S'}(\mathcal{A}(x))$.*

2.2 The black pebble game

The black pebble game is a tool originally created to study compiler operations. Given a DAG representing the dependency of intermediate values in a computation on one another, a pebbling corresponds to a strategy for allocating registers to complete the computation [Set75]. The black pebble game is formally defined as follows:

Definition 5 (Black pebble game)**.** *The black pebble game is as one player game played on a DAG $G = (V, E)$. Let $S \subseteq V$ be the nodes with in-degree zero. The goal of the black pebble game is to place pebbles on some set $T \subseteq V$ of nodes with out-degree zero. A pebble may be placed on any node v if and only if all in-nodes of v contain pebbles. A pebble may be removed from a node at any time. A pebbling strategy $\mathcal{P} = \{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_t\}$ is an ordered list of subsets of V, where $\mathcal{P}_i \subseteq V$ is the set of nodes pebbled at step i. $\mathcal{P}_0 = \emptyset$ and $T \subseteq \mathcal{P}_t$. The number of steps in $\mathcal{P}$ is t and the number of pebbles (space) is given by $\max_i |\mathcal{P}_i|$. For all $v \in (\mathcal{P}_{i+1} \setminus \mathcal{P}_i)$, we require that the in-nodes of v are in $\mathcal{P}_i$. In the sequential black pebble game, we additionally require that $|\mathcal{P}_i \sqcup \mathcal{P}_{i+1}| = 1$.*

If a dependency graph G represents some computation, then black pebbling strategies give space time trade-offs for that computation. The time for a pebbling strategy $\mathcal{P} = \{\mathcal{P}_0, \ldots, \mathcal{P}_t\}$ is the number of steps, and the total space usage is the number of pebbles. We can additionally think of the cumulative complexity as $\sum_i |\mathcal{P}_i|$ and the S'-consistent memory as the number of i where $|\mathcal{P}_i| \geq S'$. In general, the number of pebbles required for an arbitrary DAG is PSPACE-hard to approximate [DL17], but we are able

to construct explicit graphs with bounded degree that require a large number of pebbles [LT82, PTC76, RD16].

Since the pebble game is just an abstraction, if we want to use it to lower bound the space complexity of problems, we need a concrete problem enforcing the pebble game rules. It has been shown that with a random oracle $\mathcal{H}$ the problem of labeling indexed nodes in a DAG G using the following function $h : V \to \{0,1\}^\lambda$ implements the black pebble game [DNW05, ABFG14, KK14, RD16] (exact construction from [RD16]):

$$h(v_i) = \begin{cases} \mathcal{H}(i,x) & v_i \in S \\ \mathcal{H}(i, h(u_1), h(u_2), \ldots, h(u_d)) & \text{where } u_j \text{ is an in-node of } v_i \end{cases}$$

This informally follows from the fact that we need to know the labels to all in-nodes of v in order to compute $h(v)$. Unfortunately it is not clear that notions of storing space over time such as cumulative complexity are correctly captured by this problem, but there has been recent work towards proving this [DFKP13, ACK+16, AS14].

Using the above labeling problem, we can construct computations that require a large amount of space from DAGs that require a large number of pebbles. Previous works have shown that pebbling stacked supercontractors have high space complexity [LT82, PTC76], which was adopted into a memory hard function [FLW13].

2.3 Memory hard functions

Memory hard functions are a type of moderately hard function where it is easy for an honest party to compute the function but moderately hard to invert [AT17]. For a sequential algorithm the number of steps gives an upper-bound on the space used by an algorithm. As such a memory hard algorithm must also take time proportional to its space usage. This is defined formally below.

Definition 6 (from [PER09])**.** *A memory hard algorithm is one that uses $S(n)$ space and $T(n)$ operations where $S(n) = \Omega(T(n)^{1-\epsilon})$*

Definition 7. *An N-memory hard function is an $f : X \to Y$ such that for any $x \in X$, $f(x)$ can be computed in $O(poly(N))$ steps using N space, but an adversary with only $N-1$ space has no more than a $\frac{1}{|Y|}$ probability of guessing $f(x)$.*

The only known (N,N)-memory hard function relies on a random oracle and requires $O(N^2)$ time [DKW11]. There exists many more time efficient memory hard functions, but they have unproven gaps between the most space efficient known algorithms and the proven lower bounds on space complexity [DFKP13, KK14]. Some of these functions additionally allow space time trade-offs, so adversaries can circumvent the space bound by paying in additional time costs [PER09, RD16, FLW13]. This leads us to consider a more relaxed definition of memory hardness.

Definition 8 (from [RD16])**.** *An* (N,N')-memory hard function *with security parameter λ is an $f : X \to Y$ that satisfies these three criteria:*

1. non-triviality: *The size of the input is independent of N.*
2. efficiency: *An honest algorithm can compute f using at most N space and time poly(N,k).*
3. memory-hardness: *Any algorithm that uses less than N' space and computes $f(x)$ with non negligible probability must use time at least 2^λ.*

Memory hard functions generally fall into two categories depending on their memory access patterns. We will focus on data-independent memory hard functions (iMHFs), whose memory access patterns are independent of the input. Note that all memory hard functions constructed from pebbling games are data-independent.

3 Memory hard functions

3.1 Argon2i [BDK16]

Argon2i is the winner of the password hashing competition. The initial proposal of the algorithm had no proofs of memory hardness, but more recent works have proven forms of memory hardness in the random oracle model using pebbling. For simplicity we will only look at the sequential version of Argon2i.

Definition 9 (from [BZ17])**.** *An $(n, \delta, D = \{D_i\})$-random DAG is a randomly generated DAG on n nodes with maximum in-degree δ. The graph has edges (v_i, v_{i+1}) for each $i \in [0, n)$ and $\delta - 1$ random edges $(v_{r(i,1)}, v_i), \ldots, (v_{r(i,\delta-1)}, v_i)$ for each v_i, where $r(i, j) \sim D_i$.*

Definition 10 (Argon2i graph)**.** *An Argon2i-A graph is an $(N, 2, U)$-random DAG. The Argon2i-B graph is an $(N, 2, D)$-random DAG, where D_i is the distribution:*

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

While Argon2i was not initially presented with any proofs of hardness, recent work on pebbling memory hard functions have given us some bounds.

Theorem 1 (from [BCGS16])**.** *Pebbling the Argon2i-A graph with S pebbles where $16 < N' \leq S$ requires a space time complexity of:*

$$ST \geq \frac{N^2}{1536}$$

except with probability $N^2 2^{5-N'/2}$.

We defer the proof of this theorem to theorem 34 in [BCGS16], but it follows the same type of analysis as the sandwich graphs used in theorem 3 in this paper. The parallel cumulative complexity of Argon2i-A and B was lower bounded in [BZ17] as $\tilde{\Omega}(N^{\frac{5}{3}})$. This is a direct consequence of the following theorem:

Theorem 2 (from [BZ17])**.** *Let G be an (n, δ, U)-random DAG. With probability $1 - o(n^{-7})$ we have that*

$$cc^{\parallel}(G) = \tilde{\Omega}(n^{5/3})$$

While this theorem cannot be directly applied to Argon2i-B, the proof can be extended since the edges are independently sampled and $\Pr_{x \sim D_i}[x = j] = \Omega(1/i)$ [BZ17].

3.2 Balloon hashing [BCGS16]

Balloon hashing is a recent memory hard functions based on pebbling.

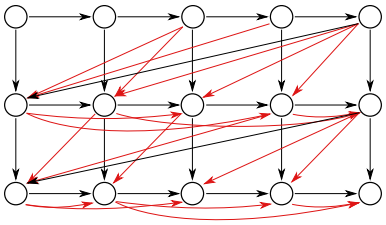

Figure 1: A balloon hash graph $G_{5,3,2}$ ($N = 5, R = 3, \delta = 2$). The black edges are fixed and the red ones are (pseudo)randomly generated. The node on the bottom right is the target.

Definition 11 (Balloon hashing graph)**.** *The Balloon hash graph* $G_{N,R,\delta} = (\{v_{n,r}\}_{n<N, r<R}, E)$ *where the nodes are divided into* R *layers of* N *nodes. There is an edge from each vertex* $v_{n,r}$ *to* $v_{n+1,r}$ *and to* $v_{n,r+1}$*. There are additional edges from each* $v_{N-1,r}$ *to* $v_{0,r+1}$*. Finally there is a set of edges* E' *where for each* $0 < n < N$ *and* $0 \le r < R$ *there are* δ *edges selected uniformly at random from the set* $\{(v_{n',r'}, v_{n,r})$ *s.t.* $((n' < n) \wedge (r' = r)) \vee ((n' > n) \wedge (r' = r - 1))\}$*. The target node is* $v_{N-1,R-1}$*.*

For concrete initialization of this function, E' is defined pseudorandomly with respect to a salt. Figure 3.2 gives an example construction of the Balloon hashing graph. In [BCGS16] they prove the following theorem about pebbling this graph:

Theorem 3 (from [BCGS16])**.** *Let* $\mathcal{A}$ *be an algorithm that pebbles node* $v_{N-1,R-1}$ *in graph* $G_{N,R,\delta=3}$*. If* $\mathcal{A}$ *uses at most* S *pebbles, then with overwhelming probability* $\mathcal{A}$ *must use space time complexity:*

$$ST(\mathcal{A}) \ge \frac{RN^2}{32}$$

When $\delta = 7$ *and* $S < N/64$*,* $\mathcal{A}$ *must use space time complexity:*

$$ST(\mathcal{A}) \ge \frac{(2^R - 1)N^2}{32}$$

Corollary 1. $G_{N,R,\delta=7}$ *defines an* $(N, \frac{N}{64})$*-memory hard function with security parameter* R*.*

This means that any attacker who tries to pebble $G_{N,R,\delta=7}$ using sublinear in N space pays an exponential cost in time. We deffer the full proof of the above theorem to the appendices of [BCGS16], but provide a bit of intuition here. Assuming there are not too many pebbles placed on the graph, it has two nice properties (with high probability) that are found in a lot of iMHF constructions:

1. Well-Spreadedness: There exists a k_δ such that for any set of k_δ adjacent nodes in the same level $V' = \{v_{n,r}, v_{n+1,r}, \ldots, v_{n+k-1,r}\}$, at least $\frac{N}{4}$ nodes on level $r-1$ are on unpebbled paths[1] to V'.

2. Expansion: All sets of k_δ nodes on each layer have unpebbled paths back to at least $2k_\delta$ vertices on the previous layer.

These properties tell us that we can fix k_δ nodes that are not currently pebbled in our graph and they must have many predecessors that cannot also be pebbled by a space efficient adversary. Thus an adversary that wants to save on space must do a large amount of recomputing. As we mentioned in section 2.2, the

[1] Paths that do not have a pebbled vertex.

problem of pebbling a graph can be instantiated with a graph labeling problem. Since the paper where it was first proposed, the analysis of balloon hashing has been tightened by viewing the graph as a stack of localized expanders [RD16]. By setting the correct parameters any efficient algorithm for balloon hash has an $\Omega(N)$-consistent memory of $\Omega(NR)$ [RD16]. The cumulative complexity of balloon hashing was lower bounded by $\tilde{\Omega}((NR)^{5/3})$ using properties of Balloon hashing that make it similar to an $(NR, \delta-1, U)$ random graph, allowing us to apply theorem 2.

3.3 Cumulative complexity attacks

Both Argon2i and Balloon hash are vulnerable to a generic attack on random DAGs described in [AB16]. Cumulative complexity allows an adversary to remain efficient when using algorithms that feature small duration spikes in memory usage.

Theorem 4 (from [AB16])**.** *Let G be an (n, δ, U)-random DAG. Then there exists an algorithm $\mathcal{A}$ that pebbles the graph with parallel cumulative complexity $cc^{\parallel} = O(n^{\frac{7}{4}}\delta \log n)$ with high probability.*

Corollary 2. *Argon2i and Balloon hash graphs with n nodes can be labeled in parallel cumulative complexity $cc^{\parallel} = O(n^{\frac{7}{4}} \log n)$.*

This algorithm relies on the observation that with high probability, random DAGs with constant indegree have are not depth robust [AB16]. This means that for a random DAG G, there exists a small set of nodes S such that the graph $G \setminus S$ has a small depth d. The algorithm proceeds in two phases. In a balloon phase up to $O(n)$ nodes of G are pebbled in a greedy fashion to place pebbles at higher depth nodes of G. In a light phase, all nodes that will not be needed for the next balloon phase, are not the highest depth node computed so far, and are not in S are unpebbled. Since we keep the nodes in S pebbled, the remaining graph $G \setminus S$ has low depth and thus pebbles removed during the light phases can be replaced in at most d rounds.

More recently [ABP17] exploited additional structure present in Argon2i-A and Balloon hashing to label these graphs with parallel cumulative complexity $\tilde{O}(n^{1+1/\sqrt{2}})$. It uses a stronger notion of non depth robustness to recursively perform the pebbling algorithm in [AB16] to pebble specific intermediate nodes. This improves on a greedy pebbling strategy featured in the balloon phase of the construction.

4 Conclusion

In this survey we looked at how to formally define data-independent memory hard functions (iMHFs) using the black pebbling game, saw two candidate iMHF constructions with bounds on different metrics of space and time complexity, and a high level picture on a general attack against iMHFs exploiting low depth robustness. Some open problems in the area include getting tighter bounds on memory hardness through either stronger attacks on existing constructions or a tighter analysis of pebbling lower bounds. Another interesting direction is analyzing the cumulative complexity of arbitrary DAGs. If we can efficiently compute the cumulative complexity of a DAG, then this will give us a good way to evaluate new iMHF candidates.

References

[AB16] Joel Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. Cryptology ePrint Archive, Report 2016/115, 2016. https://ia.cr/2016/115.

[ABFG14] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 538–557, Cham, 2014. Springer International Publishing.

[ABP17] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 3–32, Cham, 2017. Springer International Publishing.

[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 *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*, page 358–387, Berlin, Heidelberg, 2016. Springer-Verlag.

[AS14] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. Cryptology ePrint Archive, Report 2014/238, 2014. https://ia.cr/2014/238.

[AT17] Joël Alwen and Björn Tackmann. Moderately hard functions: Definition, instantiations, and applications. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 493–526, Cham, 2017. Springer International Publishing.

[BCGS16] Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. Cryptology ePrint Archive, Report 2016/027, 2016. https://ia.cr/2016/027.

[BDK16] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 292–302, 2016.

[BST98] P. Beame, M. Saks, and J.S. Thathachar. Time-space tradeoffs for branching programs. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*, pages 254–263, 1998.

[BZ17] Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of argon2i. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 445–465, Cham, 2017. Springer International Publishing.

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. Cryptology ePrint Archive, Report 2018/226, 2018. https://ia.cr/2018/226.

[DFKP13] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. Cryptology ePrint Archive, Report 2013/796, 2013. https://ia.cr/2013/796.

[DKW11] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In *TCC*, 2011.

[DL17] Erik D. Demaine and Quanquan C. Liu. Inapproximability of the standard pebble game and hard to pebble graphs. *ArXiv*, abs/1707.06343, 2017.

[DNW05] Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, pages 37–54, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[FLW13] Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena: A memory-consuming password-scrambling framework. Cryptology ePrint Archive, Report 2013/525, 2013. https://ia.cr/2013/525.

[Hel80] M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[KK14] Nikolaos P. Karvelas and Aggelos Kiayias. Efficient proofs of secure erasure. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 520–537, Cham, 2014. Springer International Publishing.

[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, oct 1982.

[PER09] COLIN PERCIVAL. Stronger key derivation via sequential memory-hard functions. 01 2009.

[PTC76] Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, page 149–160, New York, NY, USA, 1976. Association for Computing Machinery.

[RD16] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In *Proceedings, Part I, of the 14th International Conference on Theory of Cryptography - Volume 9985*, page 262–285, Berlin, Heidelberg, 2016. Springer-Verlag.

[Set75] Ravi Sethi. Complete register allocation problems. *SIAM Journal on Computing*, 4(3):226–248, 1975.