Modeling Argon 2i: Pebbling Bounds for Grid DAGs

Srinivas Arun
sarun@mit.edu

Jordan Lefkowitz
jordanll@mit.edu

Derek Liu
derekl@mit.edu

May 12, 2026

1 Introduction

A common attack model in cryptography is *precomputation*, where an adversary stores the hashes of various inputs in advance so that they can quickly invert hashed inputs later. Such attacks have become significantly cheaper due to the emergence of highly parallelized hardware. Specops, a cybersecurity company, estimates that a modern high-end GPU can calculate hundreds of billions of standard hash functions per second [8].

To defend against parallel precomputation attacks, researchers have exploited the so-called "memory wall", a phenomenon first observed by Wulf and McKee in 1994 [10]. Processor speed has improved exponentially faster than DRAM bandwidth, meaning most modern hardware is bottlenecked by memory. Thus, if hash computations have large memory requirements, then adversaries cannot take advantage of specialized hardware. This is the motivation for *memory-hard functions*, which were introduced by Percival in 2009 [9].

Argon2 is the winner of the Password Hashing Competition and one of the most widely used memory-hard functions. In this paper, we study a simplified graph-theoretic model of Argon2 based on pebbling games. Sections 2 and 3 introduce Argon2 and our model, respectively. In Section 4, we establish memory-hardness lower bounds for specific instances of the model. We then derandomize a construction of Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter [6] proving space-time lower bounds in Section 5. Finally, in Section 6, we discuss the implications of our results for the future use and parameterization of Argon2.

2 Argon2

In this section, we describe the Argon2 function introduced in [3]. In reality, Argon2 computes several "lanes" which factor into the final output; however, for ease of analysis, we focus here on a single lane.

Argon2 operates on a large fixed-size memory array of n blocks of size 1024 bytes, which it fills over T passes. We let $B^t[i]$ denote the value of the ith block after the tth pass. The initial value $B^0[0]$ is based on the parameters of the Argon2 instance. All other blocks $B^t[i]$ are computed as follows:

$$B^t[i] \leftarrow G\Big(B^t[i-1] \oplus G(B^{t'}[i'])\Big) \oplus B^{t-1}[i],$$

where

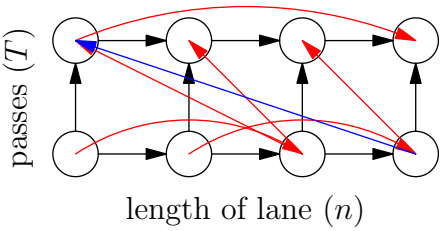
- G is a "compression" function, designed to exhibit OWF behavior,
- (t', i') are drawn from some distribution where $t' = t$ and $i' < i$, or $t' = t - 1$ and $i' > i$ (in particular, $B^{t'}[i']$ represents another block that exists at the time that $B^t[i]$ is being computed).

The output of the function is the value of the final block in the final pass, or $B^{T-1}[n-1]$. The goal of the recursive dependencies is to try to ensure that any adversary must store the entire array in order to compute the output.

Argon2 has two variants: Argon2i, in which the pseudorandom dependencies (t', i') are predetermined based on the parameters, and Argon2d, in which these dependencies depend on the data within the blocks. However, Argon2d is less commonly used because it is susceptible to side-channel attacks (one can gain information about the data based on which parts of memory are accessed). Thus, we focus on Argon2i for the remainder of this paper.

3 Pebbling Games

We treat the compression function G as a black box. We can then model Argon2i's computation dependencies by a DAG on a $T \times n$ grid, as shown below:



The t^{th} row of the grid represents B^t. The full dependency graph is made up three components:

1. The grid graph (drawn in black)
2. The wrap-around edges linking the right endpoint of each row to the left endpoint of the next (drawn in blue)
3. The pseudorandomly chosen back pointers from each vertex (drawn in red)

We can model an attacker's memory usage via the following intuitive process, called a *pebbling game*:

- The blocks that the attacker currently has in memory are represented by pebbles on vertices of the DAG.
- In one turn, the adversary can either
  - place a new pebble on any vertex whose dependencies all currently have pebbles on them, or
  - remove a pebble from any vertex.

  (In particular, the adversary can always place a pebble on the bottom left vertex, which has no dependencies).
- The adversary wins when they place a pebble on the upper right vertex of the grid.

The space an adversary requires, called the *pebbling number*, is the minimum number of pebbles an adversary needs in order to win the game. (Note that the number of pebbles used is defined as the maximum number of pebbles present at any point during the computation process.) The *space-time* an adversary requires is the minimum possible value of the product (pebbles used) $\times$ (turns taken) over all possible strategies.

Ultimately, space-time is a more relevant metric, because it controls the throughput of an adversary's hash calculations.

The simplest strategy for the adversary on our grid DAG is to compute Argon2 in the standard way: pebble the rows in order from lowest to highest, which requires $\Theta(n)$ space and $\Theta(Tn)$ time, for $\Theta(Tn^2)$ space-time. We will consider an attack successful if it uses significantly less space or space-time than the simple attack. Generally, it is known that any DAG with nT vertices and bounded degrees can be pebbled in $O(\frac{nT}{\log(nT)})$ space [7], providing a space-efficient attack when $T \leq o(\log n)$. However, such space-efficient pebbling arguments often completely disregard time.

We note that pebbling games are not a perfect model of memory hardness for several reasons:

- The function G may have structure that an attacker can exploit in order to avoid computing all dependencies.
- Even assuming that G is a black box, there exist memory-reuse schemes such as TreeEval [4] which are extremely space-efficient. However, these techniques are impractically slow in the general case.
- Even graphs with a large pebbling number may be susceptible to parallel attacks. For instance, if 99% of an adversary's pebbling steps require only 2 pebbles while the other 1% requires up to 10^9 pebbles, then with some clever memory reallocation, 99% of each hash computation can be parallelized, even though the pebbling bound is 10^9. Such "cumulative complexity" attacks are investigated in [1] and [2].

Despite these flaws, we believe that studying pebbling games is useful for understanding the behavior of Argon2 in practice. The function G that Argon2 uses is quite complex, combining bitwise operations (XOR and bitshift) with arithmetic ones (addition and multiplication), so it is difficult to find any exploitable structure in G. Thus, we believe black-box pebbling games are a reasonable model to analyze.

4 Space Bounds in Grid Pebbling Games

In this section, we show that simulating Argon2i via pebbling requires a lot of space, even without the pseudorandom auxiliary dependencies. We later observed that our proof strategy is closely related to an argument of Stephen Cook [5].

Theorem 4.1: Grid Pebbling Lower Bound

Let $G_{a,b}$ be the $a \times b$ grid DAG with a rows and b columns, where edges point up and to the right. The minimum number of pebbles needed to pebble G is exactly $\min(a, b) + 1$.

Proof. Since $G_{a,b}$ is symmetric in its horizontal and vertical dimensions, assume WLOG that $a \geq b$. Assign coordinates such that the bottom left vertex is $(1, 1)$ and the top right vertex is (b, a).

To show that $b+1$ pebbles in *sufficient*, we simply pebble $G_{a,b}$ in the same way we would pebble the Argon2i DAG. Specifically, we first pebble the bottom row. Then, iterating through the remaining $a - 1$ rows from left to right, we repeatedly place a pebble on the next vertex (x, y) and then remove the pebble at vertex $(x, y - 1)$.

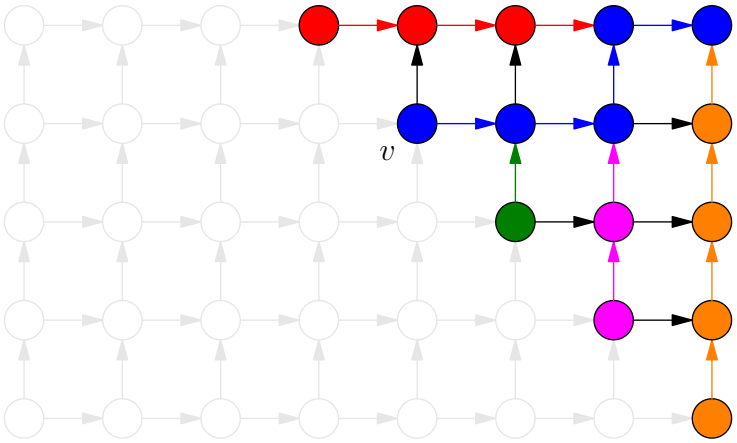
We now show that $b+1$ pebbles is *necessary*. Let D_b denote the diagonal formed by vertices (x, y) satisfying $x + y = b + 1$. Consider the last instance t in time when a pebble is paced on a vertex $v = (v_x, v_y) \in D_b$.

Let U_b be the upper triangular region of vertices (x, y) satisfying $x + y \geq b + 1$. Since no pebbles cross D_b after time t, we can restrict our focus to the region U_b.

Now, assume WLOG that the pebble placed on v at time t eventually contributes to the placement of a pebble at (b, a). That is, assume there exists an up-right path P of vertices p_0, p_1, $\ldots$, p_{a-1} along with an increasing sequence timestamps $t_0 < t_1 < \cdots < t_{a-1}$ such that:

1. $p_0 = v$ and $p_{a-1} = (b, a)$
2. $t_0 = t$
3. For every $1 \leq i \leq a - 1$, a pebble was placed on p_i at time t_i

(If such a path didn't exist, we could simply remove the pebble placed on v at time t along with all future dependent pebbles, then restart the argument.)



An example path P and partition of $U_b \setminus P$ into straight paths

Next, partition $U_b \setminus P$ into a collection $\mathcal{Q}$ of $a - 1$ straight paths, by taking the $a - v_x$ horizontal paths above P as well as the $b - v_y$ vertical paths below P. For each $1 \leq i \leq a - 1$, the two vertices leading into p_i are p_{i-1} and a second vertex q_i. At the time t_i, there must have been pebbles on both p_i and q_i. Observe that the q_i all lie on different paths in $\mathcal{Q}$, and thus exactly one q_i lies on each path. Moreover, if some path $Q \in \mathcal{Q}$ contained no pebbles at time t, then it could never contain any pebbles after time t. Hence, at t, we are guaranteed the existence of:

1. A pebble on each path $Q \in \mathcal{Q}$
2. A pebble on v by definition
3. A pebble on every vertex leading into v, since a pebble was placed on v at time t

which totals to at least $|\mathcal{Q}| + 2 = a + 1$ pebbles if v is on the edge of the grid, or $|\mathcal{Q}| + 3 = a + 2$ pebbles if v is in the interior of the grid, as desired. □

Theorem 4.1 is actually very strong. By applying the theorem to the Argon2i graph with $a = T$ and $b = n$, we know that when $T \geq n$, there is no pebbling attack of Argon2i using less space than the standard computation. Moreover, we showed this space bound by only considering the regular grid structure within the Argon2i DAG, leaving out the pseudorandom back-pointers and the wrap-around edges. In section 7, we discuss our progress in proving stronger space lower bounds for small T when some back pointer edges are reintroduced into the grid graph.

5 Space-time Bounds in Grid Pebbling Games

While it is difficult to prove meaningful lower bounds on space complexity in pebbling games, the question of space-time complexity can be much more tractable. Boneh, Corrigan-Gibbs, and Schechter [6] prove that with its pseudorandom back-pointers, Argon2i is computationally secure against pebbling attacks. Specifically, with T passes over a lane of length n, the required space-time for the corresponding pebbling game is $\Theta(Tn^2)$ with high probability.

The result we present in this section is a derandomized version of this result, which also significantly improves the constant factor on the space-time bound. We provide an explicit construction of back-pointers for which the space-time lower bound of $\Theta(Tn^2)$ is guaranteed to hold. The proof of this bound is heavily inspired by the randomized proof in [6]. The back-pointers we construct are easy to compute, making them a practical alternative to Argon2i's pseudorandom back-pointers.

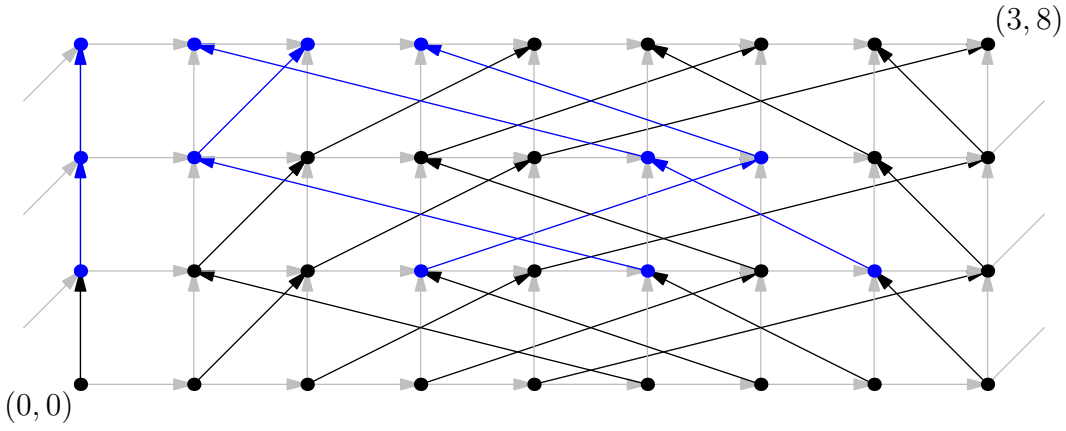
For simplicity, we will assume that $n = 2^m + 1$ is one more than a power of 2. (Note that any positive integer n is within a factor of 2 of some integer of the form $2^m + 1$, so this assumption is reasonable.) We define the directed graph $A_{t,n}$ with ordered pairs (i, j) with $0 \leq i < t$ and $0 \leq j < n$ as vertices and the following edges:

- $(i, j-1) \to (i, j)$ for all $0 \leq i < T$ and $1 \leq j < n$,
- $(i-1, n-1) \to (i, 0)$ for all $1 \leq i < T$,
- $(i-1, j) \to (i, j)$ for all $0 \leq i < T-1$ and $0 \leq j < n$, and
- $(i-1, j/2 \bmod n) \to (i, j)$ for all $0 \leq i < T-1$ and $0 \leq j < n$.

As n is odd, $j/2 \bmod n$ refers to the unique residue k modulo n for which $2k \equiv j \bmod n$.

The first two types of edges will be referred to as *sequential* edges, the third as *vertical* edges, and the fourth as *back-pointer* edges. *Row i* refers to the vertices of the form (i, j) for all j.

This graph imitates the graph for Argon with T-passes over length n, with the back-pointer edges above replacing the role of the pseudorandom back-pointers in Argon. The diagram below depicts $A_{4,9}$, with an emphasis on the back-pointer edges. (The second class of edges are broken for clarity; these connect the last vertex of each row to the first vertex of the next.)



The back-pointer edges have two properties which will form the backbone of our lower bound argument:

- These edges biject the vertices between any two consecutive rows, as 2 is invertible modulo n. This limits the usefulness of having a pebble at any single vertex.
- These edges "spread" out consecutive vertices in rows. Observe the $2^2 = 4$ consecutive blue vertices in the top row of the diagram above. Following the back-pointer edges two rows back results in the 4 blue vertices in the second row, which are nearly equally spaced out. We will rigorously prove this property shortly.

Theorem 5.1: Space-time Lower Bounds in Pebbling $A_{T,n}$

Suppose that $T \geq cm$ for some constant $c > 1$. Consider a pebbling game on the graph $A_{T,n}$. Consider a pebbling game which gets a pebble on the vertex $(T-1, n-1)$.

- If the game uses $s \geq 2^{m-1}$ space, then its space-time complexity is at least

$$s(2Tn - s) \geq 2^{m-1}(2Tn - 2^{m-1}) = Tn(n-1) - \frac{(n-1)^2}{4} \approx Tn^2.$$

- If the game uses $s < 2^{m-1}$ space, then its space-time complexity is at least

$$\frac{(c-1)^2}{4c^2} T^2(n-1)^2 \approx \frac{1}{4} T^2 n^2.$$

Proof. For ease of understanding, we start with a summary of the argument.

Summary: Split each row of the graph into n/b blocks of b consecutive vertices for a block size $b \approx 2s$. For each block B, consider the instant in time when the rightmost vertex in B is first pebbled. At this instance, the next block B' to the right is entirely unpebbled. Using the structure of the graph, we show that, *for any placement of* s pebbles on the graph at this instant in time, there still exists a large number $k(s)$ of currently unpebbled vertices throughout the graph which must be pebbled before the rightmost vertex in B' can be pebbled. Thus, the running time of the entire pebbling process is lower bounded by $\geq \frac{nT}{2s} \cdot k(s)$.

Now we give the proof in full detail. Consider the following path of sequential edges in $A_{t,n}$:

$$\begin{aligned}&(0,0) \to (0,1) \to \cdots \to (0, n-1)\\ \to &(1,0) \to (1,1) \to \cdots \to (1, n-1)\\ \to &\cdots\\ \to &(T-1,0) \to (T-1,1) \to \cdots \to (T-1, n-1).\end{aligned}$$

For the remainder of this proof, we refer to this path as a canonical ordering of vertices.

In order for a pebble to be placed on some vertex, every vertex before it must have previously received a pebble. In particular, each vertex must receive a pebble for $(T-1, n-1)$ to receive one, so the time of any pebbling game which places a pebble on $(T-1, n-1)$ is at least Tn. Furthermore, if only $s < Tn$ space is used, then at least $Tn - s$ of these pebbles must later be used, so the time is at least $2Tn - s$.

Let s be the space used in such a pebbling game. If $s \geq Tn$, then the space-time complexity is at least $(Tn)(Tn) = T^2n^2$. If $2^{m-1} \leq s < Tn$, then the space-time complexity is at least

$$s(2Tn - s) \geq 2^{m-1}(2Tn - 2^{m-1}) = Tn(n-1) - \frac{(n-1)^2}{4}.$$

Thus, it remains to prove the statement for $s < 2^{m-1}$.

Let $b = 2^a$ be the smallest power of 2 which is greater than $2s$. By assumption, $b \leq 2^m < n$. For $0 \leq i < T$ and $0 \leq k < 2^{m-a} = (n-1)/b$, let $B_{i,k}$ denote the subset of the 2^m vertices of the form (i, j) with $bk \leq j < b(k+1)$. In other words, we partition each row of $n = 2^m + 1$ vertices into 2^{m-a} blocks of $b = 2^a$ vertices, ignoring the last vertex of each row. Note that as the vertices are ordered, so are the blocks, in the order $B_{0,0}, \ldots, B_{0,2^{m-a}-1}, B_{1,0}, \ldots, B_{1,2^{m-a}-1}$, and so on.

Let $T_{i,k}$ denote the first time in the pebbling game when the last vertex of block $B_{i,k}$ (i.e., $(i, b(k+1)-1)$) receives a pebble. At this point, no vertex in any block after $B_{i,k}$ could have ever received a pebble, as all such vertices are after $(i, b(k+1)-1)$.

Suppose $i \geq a$ and $k > 0$ (i.e., $T_{i,k}$ is not the first block in its row). We will prove that at least $O((i-a)n)$ time must occur between $T_{i,k-1}$ and $T_{i,k}$.

As the back-pointer edges biject the vertices in any two adjacent rows, for each vertex (i, j) in row $i \geq a$, there is a unique path of a back-pointer edges with ends at (i, j). This path starts at vertex $(i-a, j/2^a \bmod n)$; let p_j denote this path and $c_j = j/2^a \bmod n$.

We will prove that for $bk \leq j < b(k+1)$ (i.e., $(i, j) \in B_{i,k}$), no two of the indices c_j differ by less than 2^{m-a}. Indeed, if $|c_j - c_{j'}| = d$ is strictly between 0 and 2^{m-a}, then

$$2^a \leq |2^a c_j - 2^a c_{j'}| \leq (2^{m-a} - 1)2^a = n - (2^a + 1).$$

Since $2^a c_j \equiv j \bmod n$ and $2^a c_{j'} \equiv j' \bmod n$, we know

$$2^a c_j - 2^a c_{j'} \equiv j - j' \pmod{n},$$

so j and j' differ by at least $\min(2^a, 2^a + 1) = 2^a = b$. This cannot be true if (i, j) and (i, j') are in the same block (unless $j = j'$).

(In fact, $\lfloor n/b \rfloor = 2^{m-a}$, so the b indices c_j are essentially as "spaced out" as possible.)

If $c_j \geq 2^{m-a} - 1$, let $I_j = \{(i-a, c_j - 2^{m-a} + 1), (i-a, c_j - 2^{m-a} + 2), \ldots, (i-a, c_j - 1), (i-a, c_j)\}$, and let J_j denote the union of the vertices in I_j and p_j. Because of sequential edges, the vertices in I_j form a path ending at $(i-a, c_j)$, so the vertices of J_j form a path ending at (i, j).

As shown above, the subsets I_j are all disjoint. The paths p_j are also disjoint (as back-pointers biject adjacent rows), so the subsets J_j are as well. The above also shows that at most one value of j with $bk \leq j < b(k+1)$ has $c_j < 2^{m-a}$, so this interval is defined for at least $b - 1$ values of j.

At most s pebbles are present at any time, so at time $T_{i,k-1}$, at most s of the paths J_j contain one or more pebbles. Thus, at least $(b-1) - s \geq b/2 = 2^{a-1}$ of these paths don't have any pebbles at time $T_{i,k-1}$.

Recall that at time $T_{i,k-1}$, no pebbles have been placed on any vertex of block $B_{i,k}$. Thus, to reach $T_{i,j}$, each vertex of $B_{i,k}$ must receive a pebble. If path J_j has no pebbles at time $T_{i,k-1}$, then (i, j) can only receive a pebble if every vertex in path J_j receives one. In particular, J_j contains I_j and thus at 2^{m-a} vertices in row $i - a$, all of which must receive pebbles between times $T_{i,k-1}$ and $T_{i,k}$. Thus, between these times, at least

$$2^{a-1} \cdot 2^{m-a} = 2^{m-1}$$

vertices in row $i - a$ must receive pebbles.

If $i \geq a+1$, then a similar argument applies in row $i-(a+1)$. Indeed, there are vertical edges $(i-(a+1), c_j) \to (i-a, c_j)$, allowing us to move the sets I_j into row $i-(a+1)$ while maintaining valid paths. Likewise, this

argument applies in every row between 0 and $i-a$. Thus, the number of pebbles that must receive vertices between times $T_{i,k-1}$ and $T_{i,k}$ is at least

$$(i-a+1)2^{m-1},$$

which is also a lower bound for the time between $T_{i,k-1}$ and $T_{i,k}$.

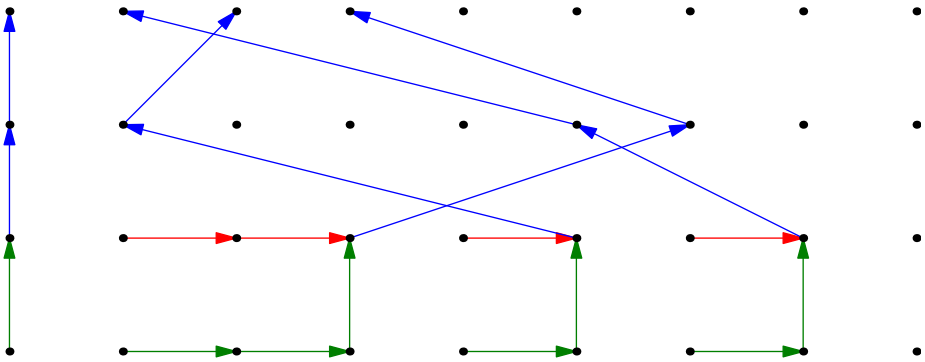
This argument also applies for $T_{i-1,2^{m-a}-1}$ and $T_{i,0}$ (even if $i=a$). Summing over all i and k with $i \geq a$, we conclude the amount of time the pebbling game must take is at least

$$\sum_{i=a}^{T} \sum_{k=0}^{2^{m-a}-1} (i-a+1)2^{m-1} = \frac{(T-a)(T-a+1)}{2} \cdot 2^{m-a} \cdot 2^{m-1}.$$

Note that $a \leq m$, so $T - a \geq \frac{c-1}{c} \cdot T$. Also, $s \geq 2^{a-2}$, as otherwise b would have been smaller. Thus, the space-time complexity is at least

$$s \cdot \frac{(T-a)(T-a+1)}{2} \cdot 2^{m-a} \cdot 2^{m-1} \geq 2^a \cdot \left(\frac{c-1}{c}\right)^2 T^2 \cdot 2^{2m-a-2} = \frac{(c-1)^2}{4c^2} T^2 (n-1)^2. \qquad \square$$

The following diagram visualizes this argument on $A_{4,9}$ with $b = 2^2$ on the top-left block $B_{3,0}$. Only relevant edges are shown; blue edges are paths p_j, red edges are paths I_j (in row $3-2=1$), and green edges show the argument in lower rows.



Thus, the pebbling game on $A_{T,n}$ has a space-time complexity of at least $\Theta(tn^2)$, and any attempt to use $s < 2^{m-1} = (n-1)/2$ space incurs an additional space-time penalty of a factor of at least $\Theta(T)$.

Unfortunately, the graph $A_{T,n}$ cannot quite be pebbled in the same way Argon works. This graph has edges of the form $(i-1, j_1) \to (i, j_2)$ with $j_1 < j_2$, i.e., a vertex can depend on a vertex in the previous row with an earlier vertex. Argon's method of pebbling would replace the pebble on $(i-1, j_1)$ with one on (i, j_1) before reaching (i, j_2).

Instead, we provide the following pebbling game on $A_{T,n}$ which uses $\frac{5}{4}n + O(1)$ space (assuming $n \geq 5$).

1. Place pabbles on the entire first row in order.
2. For $i = 1, 2, \ldots, T-1$:
   (a) At this point, the entirety of row $i-1$ has pebbles, and no other row has pebbles.
   (b) Place a pebble on $(i, 0)$, then remove the pebble from $(i-1, 0)$.
   (c) For $j = 1, 2, \ldots, 2^{m-2}$:

i. Place a pebble on $(i, 2j-1)$ and a pebble on $(i, 2j)$, then remove the pebble from $(i-1, j)$.

(d) For $j = 2^{m-1}+1,\ 2^{m-1}+2,\ \ldots,\ n-1$:

i. Place a pebble on (i, j), then remove the pebble from $(i-1, j)$.

(e) At this point, all of row i has pebbles. Remove any remaining pebbles from row $i-1$.

We check that all pebble placements are valid.

- Step 2b is always valid as $(i-1, 0)$ and $(i-1, n-1)$ both have pebbles.
- Throughout step 2c, no pebbles are ever removed from (i, k) for $k > 2^m$. Also, $2j-1 \geq j$, so at iteration j of step 2c, there are pebbles on $(i, 2j-2)$, $(i-1, 2j-1)$, and $(i-1, j+2^{m-1})$. Thus, the placement of a pebble on $(i, 2j-1)$ is valid.

  After this, $(i, 2j-1)$, $(i-1, 2j)$, and $(i-1, j)$ all have pebbles, so the placement of a pebble on $(i, 2j)$ is valid.
- If $j > 2^{m-1}$, then $j/2 \bmod n$ is greater than 2^{m-2}, so Step 2c did not remove the pebbles at $(i-1, j/2 \bmod n)$ or $(i-1, j)$. Thus, Step 2d is valid.

At step 2b, there are always n pebbles (or temporarily $n+1$). Each iteration of step 2c increases the number of pebbles by 1, so during the last iteration of step 2c, the most number of pebbles there are at any time is $n + (2^{m-2}-1) + 2 = n + 2^{m-2} + 1 = \frac{5n+3}{4}$, and this iteration ends with $\frac{5n-1}{4}$ pebbles. Step 2d never increases the number of pebbles, so the space of this pebbling game is $\frac{5n+3}{4}$.

Each vertex receives a pebble exactly once and loses a pebble at most once, so this pebbling game takes at most $2Tn$ steps. Thus, the space-time complexity is $2Tn \cdot \frac{5n+3}{4} n \approx \frac{5}{2} Tn^2$.

The previous theorem implies that as long as c is large enough, no pebbling attack will ever achieve a space-time complexity less than $\frac{2}{5}$ of our pebbling game. Furthermore, any attempt to use less than $\frac{2}{5}$ of its space ($\frac{2}{5} \cdot \frac{5}{4} n \approx 2^{m-1}$) suffers an additional space-time penalty of a factor of approximately $T/4$. Note that $c \geq 6$ and $n \geq 33$ ensures

$$\frac{(c-1)^2}{4c^2} T^2 (n-1)^2 \geq \frac{(c-1)^2}{4c} \cdot \frac{n-1}{n} Tn(n-1) \geq \frac{5^2}{4 \cdot 6} \cdot \frac{32}{33} Tn^2 > Tn^2,$$

and hence attacks using less than 2^{m-1} space require higher space-time complexity than attacks with at least 2^{m-1} space.

Thus, the graph $A_{T,n}$ provides a viable alternative to Argon2i which uses predetermined back-pointers, guaranteeing memory-hardness (up to a factor of $\frac{2}{5}$) against all pebbling attacks with $n \geq 33$ and $T \geq 6 \log_2 n$, with larger values of T yielding stronger guarantees against low-memory attacks. As T can be as small as $\Theta(\log n)$, this is a much more viable alternative than the suggestion of $T = n$ in Section 4.

6 Implications and Recommendations

The most interesting feature of Argon2i is its use of pseudorandom back-pointers. Intuitively, these dependencies increase the complexity of the computation DAG, suggesting that they should be essential to Argon2i's security. However, our results provide evidence to the contrary.

- Theorem 4.1 suggests that when using $T = \Theta(n)$ passes, the additional back-pointers are entirely unnecessary: strong memory hardness can already be proven without them. Eliminating the back-pointers could simplify the implementation, although the required number of passes may be impractically large.

- Theorem 5.1 further suggests that even with only $T = \Theta(\log n)$ passes, the extra back-pointers can be derandomized while preserving equally strong space–time bounds. A fixed construction could enable hardware optimizations that improve hashing efficiency for honest users, while still preventing efficient precomputation attacks by parallel adversaries.

However, as mentioned earlier, the pebbling game model is only an approximation of real-world attacks, so these conclusions should be interpreted with appropriate caution.

7 Future Work

One potentially interesting direction for future work is studying pebbling games on grid DAGs with *fixed* back-pointers. Let $G_{T,n,k}$ be the DAG on a $T \times n$ grid where each vertex has edges from the vertex to its left, the vertex below it, and the vertex exactly k to the left. Then there exist attacks using space $n+1$ (by traversing rows from bottom to top) and space $kT+1$ (by traversing columns from left to right).

We can calculate the exact pebbling number for small cases by simply performing a breadth-first-search over all sets of pebbles of a certain size. For $k = 3$, we obtain:

T,n	6	7	8	9	10	11	12	13	14
2	5	6	6	6	7	7	7	7	7
3	6	7	8	8	9	9	9	10	10
4	7	8	9	9					

and for $k = 4$:

T,n	8	9	10	11	12	13	14
2	5	6	6	6	6	7	7
3	6	7	8	8	8	9	
4	7	8	9				

We conjecture that the pebbling number eventually stabilizes to either $n+1$ or $kT+1$ for all such graphs, as corroborated by the highlighted values in the $k = 3$ table. However, in our experience, graphs with fixed back-pointers are much more difficult to analyze than graphs with no back-pointers.

Another interesting direction is analyzing the gap between pebbling games and true attacks. This includes determining the extent to which space-efficient attacks like TreeEval can be made time-efficient, and identifying weaknesses in the compression function G used in Argon2.

8 Acknolwedgements

We would like to sincerely thank the 6.5610 course staff for teaching us about cryptographic primitives and threat models throughout the semester. We are especially grateful to our TA, Kevin He, for introducing us to Argon2 and for the many insightful discussions that helped shape the direction of this work.

9 Contributions

As this project was entirely theoretical, we proved all of our results through group discussions during our in-person meetings.

References

[1] Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions, March 2016.

[2] Joel F. Alwen, Jeremiah Blocki, and Krzysztof Z. Pietrzak. Depth-robust graphs and their cumulative memory complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, volume 10212 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2017.

[3] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: The memory-hard function for password hashing and other applications, December 2015. Version 1.2.1.

[4] James Cook and Ian Mertz. Tree evaluation is in space O(log n · log log n). In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC '24, pages 1268–1278, New York, NY, USA, 2024. Association for Computing Machinery.

[5] Stephen A. Cook. An observation on time-storage trade off. *Journal of Computer and System Sciences*, 9(3):308–316, December 1974.

[6] Henry Corrigan-Gibbs, Dan Boneh, and Stuart Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In *Advances in Cryptology – ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 220–248. Springer, 2016.

[7] John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant. On time versus space. *Journal of the ACM*, 24(2):332–337, April 1977.

[8] David Ketler. Do ai gpus make password cracking faster? testing nvidia h200 and amd mi300x, March 2026.

[9] Colin Percival. Stronger key derivation via sequential memory-hard functions. In *BSDCan 2009*, May 2009.

[10] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, March 1995.