2017 IEEE European Symposium on Security and Privacy

Towards Practical Attacks on Argon2i and Balloon Hashing

Joël Alwen
IST Austria

Jeremiah Blocki
Purdue University

Abstract—**The algorithm Argon2i-B of Biryukov, Dinu and Khovratovich is currently being considered by the IRTF (Internet Research Task Force) as a new de-facto standard for password hashing. An older version (Argon2i-A) of the same algorithm was chosen as the winner of the recent Password Hashing Competition. An important competitor to Argon2i-B is the recently introduced Balloon Hashing (BH) algorithm of Corrigan-Gibs, Boneh and Schechter.**

A key security desiderata for any such algorithm is that evaluating it (even using a custom device) requires a large amount of memory amortized across multiple instances. Alwen and Blocki (CRYPTO 2016) introduced a class of theoretical attacks against Argon2i-A and BH. While these attacks yield large asymptotic reductions in the amount of memory, it was not, a priori, clear if (1) they can be extended to the newer Argon2i-B, (2) the attacks are effective on any algorithm for practical parameter ranges (e.g., 1GB of memory) and (3) if they can be effectively instantiated against any algorithm under realistic hardware constrains.

In this work we answer all three of these questions in the affirmative for all three algorithms. This is also the first work to analyze the security of Argon2i-B. In more detail, we extend the theoretical attacks of Alwen and Blocki (CRYPTO 2016) to the recent Argon2i-B proposal demonstrating severe asymptotic deficiencies in its security. Next we introduce several novel heuristics for improving the attack's concrete memory efficiency even when on-chip memory bandwidth is bounded. We then simulate our attacks on randomly sampled Argon2i-A, Argon2i-B and BH instances and measure the resulting memory consumption for various practical parameter ranges and for a variety of upperbounds on the amount of parallelism available to the attacker. Finally we describe, implement, and test a new heuristic for applying the Alwen-Blocki attack to functions employing a technique developed by Corrigan-Gibs et al. for improving concrete security of memory-hard functions.

We analyze the collected data and show the effects various parameters have on the memory consumption of the attack. In particular, we can draw several interesting conclusions about the level of security provided by these functions.

- **For the Alwen-Blocki attack to fail against practical memory parameters, Argon2i-B must be instantiated with more than 10 passes on memory — beyond the "paranoid" parameter setting in the current IRTF proposal.**
- **The technique of Corrigan-Gibs for improving security can also be overcome by the Alwen-Blocki attack under realistic hardware constraints.**
- **On a positive note, both the asymptotic and concrete security of Argon2i-B seem to improve on that of Argon2i-A.**

1. Introduction

The goal of key-stretching is to protect low-entropy secrets (e.g., passwords) against brute-force attacks. A good key-stretching algorithm should satisfy the properties that (1) an honest party can compute a single instance of the algorithm on standard hardware for a moderate cost, (2) the amortized cost of computing the algorithm on multiple instances on customized hardware is not (significantly) reduced. The first property ensures that it is possible for honest parties (who already know the secret) to execute the algorithm, while the later property ensures that it is infeasible for an adversary to execute a brute-force attack with millions/billions of different guesses for the user's secret. Key-stretching techniques like hash iteration (e.g., `bcrypt`) fail to achieve the later property as the cost of evaluating hash functions like SHA256 can be dramatically reduced by building Application Specific Integrated Circuits (ASICs).

Memory hard functions (MHFs), first explicitly introduced by Percival [Per09], are a promising key-stretching tool for achieving property two. In particular, MHFs are motivated by the observation that the cost of storing/retrieving items from memory is relatively constant across different computer architectures. Data-Independent Memory Hard Functions (iMHFs) are an important variant of MHFs due to their greater resis-

© 2017, Joël Alwen. Under license to IEEE.
DOI 10.1109/EuroSP.2017.47

IEEE computer society

tance to side-channel attacks[1] making them the recommended type of MHF for password hashing. Indeed, most of the entrants to recent Password Hashing Competition [PHC] which had the stated aim of finding a new password hashing algorithm, claimed some form of memory-hardness.

Finally, to accommodate a variety of devices and applications an MHF is equipped with a "memory parameter" (and sometimes also a "timing parameter") which fix the amount of memory (and computation, respectively) used by the honest evaluation algorithm. Ideally, the necessary amortized memory (and computation) of an adversary should also scale linearly in these parameters. Thus they can be thought of as natural security parameter(s) for the MHF.

In this work we focus on three of the most prominent iMHFs from the literature:

1) The winner of the PHC, which we call Argon2-A [BDK15].
2) A significantly updated version which we refer to as Argon2i-B [BDKJ16] which is currently being considered by the Cryptography Form Research Group (CFRG) of the IRTF as their first proposal of an MHF for wider use in Internet protocols.
3) A prominent competitor algorithm to Argon2 called Balloon Hashing (BH) [BCGS16].

We remark that an important part of IRTF proposal is a recommendation for how to choose the parameters when instantiating Argon2. In particular, it describes how to select a memory parameter σ and timing parameter τ with the effect that the honest algorithm for evaluating Argon2i-B builds a table of size σ in memory and iterates over it τ times.

1.1. Attacking an MHF

In the context of an MHF an "attack" is an evaluation algorithm with lower (possibly amortized) complexity then the honest evaluation algorithm. The quality of such an attack is the ratio between the honest algorithms complexity on a single instance and the attack's (amortized) complexity. We refer to an evaluation algorithm with quality greater than 1 as an *attack*.

Historically, the "complexity" of an attack has referred to the product of the total runtime and the largest amount of memory used at any given point during the execution as this often believed to be a good estimate of the AT (Area x Time) complexity of implementing the execution in hardware [Per09], [AS15], [BK15] which in turn provides an estimate for the cost of *constructing* the hardware [Tho79]. In this work, we have instead measured the energy complexity [AB16] of an execution which approximates the energy consumed by the attack. As discussed in [AB16] this approximates the *running cost* of hardware implementing the execution.[2] Never-the-less, for the class of attacks considered in this work, energy complexity also tightly approximates the (amortized) AT complexity of the attack. [AB16].

1.1.1. Argon2 History. The Argon2 specification [BDK16] has already undergone 4 reversions since its first publication with, at times, non-trivial changes to the underlying graph structure. Unfortunately any of these versions are regularly referred to simply as "Argon2" with out further specification. In particular, between Argon2i-A (Version 1 in [BDK16]) and Argon2i-B (Version 1.3 in [BDK16]) the edge distribution (which describes how intermediary values relate to each) has been altered twice. However the edge distribution is probably the single most important feature of any MHF in determining its memory-hardness, i.e. its security. In particular, altering the edge distribution of an MHF can make the difference between optimal security and being completely broken. Thus, we have made an effort to distinguish between the original edge structures in the PHC Argon2i-A and the edge structure used in the IRTF proposal Argon2i-B.

To the best of our knowledge there are three known attacks against Argon2-A and none on Argon2-B. While the first attack was against Argon2-A [BK15] the two more recent ones took aim at Argon2i-A [BCGS16], [AB16].[3]

1.1.2. Balloon Hashing History. One of the most important alternative proposals to Argon2i is the Balloon Hashing (BH) algorithm of [BCGS16]. To date the only known attack on BH is [AB16].[4] Another interesting contribution of the same authors appeared in an earlier version of that work [CGBS16] where they introduced a new technique for constructing iMHFs which we refer

1. Standard MHFs (e.g., Argon2d [BDK16], `scrypt` [Per09]) are potentially vulnerable to security and privacy compromises due to cache-timing attacks [Ber], [FLW13].

2. While the construction cost of the hardware is a one-off cost which can be amortized across all evaluations ever performed on the device, the running cost is a recurring charge per instance and so seems at least as important from the point of view of an attacker evaluating the effectiveness of the attack.

3. Due to the frequent naming collision between versions it not possible to unambiguously determine the precise version considered in these attacks. However all results seem to have appeared before the newest edge structure, used in Argon2-B, was published [BDK16] and definitely before the IRTF proposal was made [BDKJ16]. Thus the history presented here reflects the best guess of the authors based on the dates of revisions and when the different attacks were published.

4. Roughly, BH is a special case of the "random graph" family of iMHFs considered in [AB16] where indegree $\delta = 3$.

to as the XOR-and-high-indegree method. They showed how it could be used to overcome the new attack on Argon2i-A described in the same work. The technique was also instantiated in the BH-Double-Buffer (BH-DB) iMHF of [CGBS16].

1.2. How Practical is AB16?

While there seems to be less debate about the effectiveness and practical consequences of [BK15], [BCGS16] the same can not be said for [AB16]. On the one hand the authors of [AB16] proved that the asymptotic complexity of their attack is far lower then that of the honest evaluation algorithm as a function of the memory and timing parameters (not solely for Argon2i-A but also for the other candidate iMHFs including a precursor to BH, BH-DB and others). In other words, the quality of their attack grows quickly in the natural security parameters of Argon2i-A (and BH) indicating severe problems with those constructions from a theoretical perspective.

However, as observed by the Argon2 authors [BDK16], $\tau \geq 4$ passes on memory seems to suffice to thwart this attack in practice (e.g., even if the memory parameter is as large as $\sigma = 2^{20}$).[5] Moreover it has been observed that [Kho16], a straightforward implementation of the [AB16] attack in hardware would require several times the amount of memory bandwidth currently possible with modern technology. Thus it may seem, as claimed by the Argon2 authors [BDK16] and others [Aum16], that the [AB16] attack does not present a threat to the real world security of Argon2-B (or BH). Indeed, in contrast to the attacks of [BK15], [BCGS16] the [AB16] attack is omitted from the security analysis in the IRTF proposal [BDKJ16].

However, despite these observations, there may yet be reasons for concern that [AB16] could behave far better in practice.

1) Due to the rigors imposed by proving statements, in [AB16] several pessimistic assumptions (detailed in Section 6) were made potentially resulting in far worse cost estimates for their algorithm then might be exhibited in any actual instantiation.
2) The algorithm of [AB16] is equipped with a variety of variables and parameters. Due to the focus on asymptotics no effort was made to optimize them. Instead asymptotically optimal values were used. However for any given concrete instance of Argon2i it is likely that the asymptotic optimal settings do not actually result in the lowest complexity of the resulting evaluation algorithm.
3) The authors did not attempt to investigate any heuristics for improving concrete complexity. Indeed, many such heuristics are far easier to implement (and test) then to analyze in theory making them bad candidates for that work but potentially still a concern in practice.
4) No work was done to examine the behavior of the attack in models of bounded (e.g. practical) parallelism or memory bandwidth.

In fact, it is perhaps somewhat surprising (not to mention disconcerting) that despite all of these omissions and pessimistic assumptions such undesirable asymptotics where still displayed against both Argon2i-A and BH.

Ultimately we are left with a rather incomplete understanding of the practicality of [AB16], especially with respect to Argon2i-B and the CFRG proposal (not to mention the other iMHFs considered in [AB16]).

1.3. Our Contribution

In this work we attempt to make progress on this front. The results in this work can be summarized as follows:

- We analyze the asymptotic complexity of the Alwen-Blocki attack [AB16] when applied to Argon2i-B demonstrating its strong asymptotic effectiveness.
- We introduce two definitive improvements to the attack of [AB16], which apply to Argon2i-A, Argon2i-B and BH. The first improvement reduces the size of the depth-reducing set S in the attack. Here, the depth-reducing set is a set S of nodes such that by removing these nodes from G, the a directed-acyclic graph (DAG) representing data-dependencies between the memory blocks produced during the evaluation of the iMHF, the depth of the resulting DAG is small. The second improvement reduces the number of 'expensive' steps necessary to execute the attack.
- We give new details about the resources (e.g., bandwidth, #cores) necessary to implement the attack of [AB16] in a custom device. This helps determine for which parameter spaces the attack is feasible using modern fabrication technology.
- We implement a simulation of the [AB16] attack in C for the case of Argon2i-A, Argon2i-B and BH together with several new heuristics for decreasing its concrete complexity.
- We implement two methods optimizing the parameters and internal variables of [AB16] so as

5. Increasing the number of passes τ slows down the evaluation of an iMHF without increasing the memory required to evaluate it.

to obtain minimal complexity for given target σ and τ. The first method may utilize arbitrary parallelism while the second method aims to minimize attack costs subject to an upper-bound on the available parallelism.

- We measure the resulting complexity of the simulation for a variety of practical parameter ranges and bounds on parallelism and we describe and analyze these results. In particular we highlight some several concerns with the parameter choices in the CFRG proposal. We also highlight several new questions spurred by our results.

2. Preliminaries

We begin by establishing some notation followed by a brief review of Argon2i-A, Argon2i-B, BH and the evaluation algorithm of [AB16].

All three iMHFs can be viewed as a modes of operation over a hash function. We use the language of "graph labeling" to describe the functions (as in [DKW11], [AS15], [AB16] for example). That is for a given memory and time parameters σ and τ respectively each iMHF is given by a directed acyclic graph $G_{\sigma,\tau} = (V, E)$ on $n = \sigma * \tau$ nodes. We number the nodes according to $V = \{1, 2, \ldots, n\} = [n]$ and for $a \leq b$ we denote the interval $\{a, a+1, \ldots, b\}$ with $[a, b]$. Each node represents an intermediary value in the computation of the iMHF. For a given input we refer to such a (k-bit) value as the "label" of the node. By deriving a random string from public parameters we effectively fix such a graph $G_{\sigma,\tau}$ from a particular distribution characterizing the iMHF. For a node v we denote by $\mathsf{parents}(v)$ the set of nodes with an edge leading to u. More generally for $S \subseteq V$ we write $\mathsf{parents}(S)$ for the set of nodes with an edge leading to a node in S.

2.1. The iMHFs

All three algorithms support an extra parallelism integer parameter $p > 0$ in order to better support honest users with multi-core machines. We now describe the case with $p = 1$. For a discussion for the more general case we refer to section Section 6.8.

In all three cases, all nodes in $V = [n]$ are initially connected by a path $\{(i, i+1) : i \in [n-1]\}$ running through the entire graph. For Argon2i-A, each node $v \in V \setminus \{1\}$ receives an additional incoming edge from a uniform random and independently chosen predecessor $u \leftarrow [\max\{v - \sigma, 1\}, v - 1]$. Similarly, for BH we add 2 such uniform and independently chosen random edges.

In the case of Argon2i-B the distribution of the random edges is somewhat more complicated. However, for the purpose of this work, it suffices that the following property holds (which can be easily verified from the specification [BDK16]). For any node $j \leq \sigma$ and $c \geq 1$ we have that

$$\Pr\left[\mathsf{parents}(j) \in [j - j/c, j-1]\right] \propto 1/\sqrt{c}$$

while for $j > \sigma$ and $c \geq 1$ we have that

$$\Pr\left[\mathsf{parents}(j) \in [j - \sigma/c, j-1]\right] \propto 1/\sqrt{c}.$$

Given a fixed graph and input we can now compute the corresponding iMHF. First the input (password, salt, etc) is hashed once to produce the label of node 1. Each subsequent label is computed by applying the hash function to the labels of the parents of the node. The final output of the iMHF is then obtained from the label of node n.

For further details on each of these algorithms we refer the interested reader to the original specifications.

2.2. The AB16 Algorithm

We describe an evaluation algorithm for an iMHF using the language of graph pebbling [HP70], [DKW11], [AB16]. Placing a pebble on a node denotes the act of computing the "label" of v. That is the intermediary value represented by the node v which is computed by applying the hash function to the intermediary values of all nodes with outgoing edges leading to v. Keeping a pebble on node v at iteration i denotes storing the label of v in memory at step i. Conversely, removing a pebble from a node denotes freeing the corresponding memory location. Clearly a pebble can only be placed at an iteration i if all parent nodes of v already have a pebble on them at the end of iteration $i-1$.[6] Since we are considering evaluations in a parallel model of computation we allow for multiple pebbles to be placed simultaneously as long as each node receiving a pebble already has all its parents pebbled at the beginning of that iteration. In a model with an upperbound of $U \in \mathbb{N}$ on parallelism we only permit up to U pebbles to be placed simultaneously in any given iteration.

Using this language, an evaluation algorithm for Argon2i is given by a *pebbling* of $G_{\sigma,\tau}$, that is a sequence $P = (P_0, P_1, P_2, \ldots, P_z)$ of subsets of V (denoting which nodes contain a pebble at the end of each iteration) such that every:

1) $P_0 = \emptyset$
2) $\forall v \in P_i \setminus P_{i-1}$ it holds that $\mathsf{parents}(v) \in P_{i-1}$
3) $n \in P_z$.

To determine the quality of an evaluation algorithm we must establish a complexity measure. For this we use

6. Indeed, an intermediary value can only be computed if all the necessary inputs to the hash function are already stored in memory.

the *energy (pebbling) complexity (EC)* [AS15] which is parametrized by the core-memory energy ratio R. This is the ratio between the cost of evaluating one call to the hash function and storing one label for an equivalent amount of time.[7] For a given ratio R, the EC is defined to be

$$\mathsf{ec}_R(P) := \sum_{i\in[z]} |P_i| + R * |P_i \setminus P_{i-1}|.$$

Intuitively it captures the total amount of energy consumed (to store memory and evaluate the hash function) during the execution. As shown in [AS15] EC scales linearly in the number of instances being evaluated. Moreover, in the case of the [AB16] attack, it also turns out to be very close to the amortized AT complexity of the algorithm. In particular, when computing several instances in parallel, it is easy to implement [AB16] such that essentially all memory cells and all hash function circuits are almost always in use.[8]

2.2.1. Quality of an Algorithm. In order to evaluate the quality of an evaluation algorithm (and in particular to determine if it is an attack) we compare to the EC of the honest (reference) algorithm $\mathcal{N}$ (e.g. the one proposed in [BDK16]). The quality of an attack $\mathcal{A}$ is given by the ratio

$$\mathsf{E\text{-}quality}(\mathcal{A}) = \frac{\mathsf{ec}_R(\mathcal{N})}{\mathsf{ec}_R(\mathcal{A})} \ .$$

Fortunately, for all three algorithms we consider, the honest algorithm is quite simple. It computes one label at a time storing them in a table of σ values. It iterates over the table τ times updating the values as it passes over them. Thus, in each case the final EC is $\mathsf{ec}_R(\mathcal{N}) = \sigma^2(\tau-1) + \sum_{i\in[\sigma]} i + R * \tau\sigma$. Ideally, a secure iMHF should have $\mathsf{E\text{-}quality}(\mathcal{A})$ as small as possible for any attacker $\mathcal{A}$ and $\mathsf{ec}_R(\mathcal{N})$ as large as possible. The first constraint ensures that an attacker cannot do (much) better than running the naive algorithm and the second constraint ensures that it is prohibitively expensive for the attacker to execute the naive evaluation algorithm millions or billions of times.

2.2.2. The AB16 Strategy. The evaluation algorithm of [AB16] is parametrized by a node set $S \subset V$ and integer $g \geq d$ where $d = \mathsf{depth}(G_{\sigma,\tau} - S)$ is the number of edges in the longest (directed) path in the graph obtained by removing S (and incident edges) from $G_{\sigma,\tau}$. The algorithm consists of two main subroutines; the *light phase* and *balloon phase*. Each light phase lasts for g iterations and upon completion a new light phase is immediately started. During the final d iterations of each light phase the algorithm also executes a balloon phase in parallel.

Intuitively the purpose of the jth light phase is to pebble target nodes $T_j = [(j-1)g+1, jg] \subseteq V$ in sequence one iteration at a time. That is, in any given iteration i, only a single pebble is placed due to the light phase, namely on node $i \in T_j$. Let node set $R_{j,i} = \mathsf{parents}(T_j \cap [i,n])$. In iteration t during the jth light phase all pebbles are removed from the graph except from nodes in the set S and in $R_{j,t+1}$. To see why all pebbles placed during a light phase are done so only when their parents already contain pebbles we consider property $\mathcal{P}_j$ which holds if at the end of the time step just before the jth light phase begins (i.e at time $(j-1)g$) all nodes in $u \leq (j-1)g$ with $u \in R_{j,1}$ contain a pebble. Notice that if $\mathcal{P}_j$ holds then any node pebbled in the jth light phase is guaranteed to have their parents pebbled. Either parent $p < v$ is such that $p \leq (j-1)g$ in which case they are pebbled during all of that light phase until v is pebbled. Or $p \in [(j-1)g, v-1]$ in which case it is pebbled during the light phase. Either way, once p contains a pebble, the pebble is never removed (at least) until v is pebbled.

Trivially property $\mathcal{P}_\infty$ holds. So it remains only to ensure that $\forall j > 1$ property $\mathcal{P}_j$ holds. This is done by running a balloon phase in parallel with the final d steps of each light phase. That is, setting $j' = jg - d + 1$, the jth balloon phase runs during interval $[j', jg]$ During these steps the algorithm never removes any pebbles from the graph. Moreover at each step it greedily pebbles any node it possibly can. We claim that that at the end of iteration jg all nodes in $[jg] \subset V$ contain a pebble. (If this is true it is trivial to ensure $\mathcal{P}_j$ simply by removing all pebbles except those in (S and) $R_{j,1}$.) To see why recall that pebbles are never removed from the set S. So at the end of iteration $j'-1$ all nodes in $S \cap [j'-1]$ are pebbled. But that means that $G' - (S \cap [j'-1])$ contains no path longer than d (where G' is the graph obtained from G by removing all nodes except those in $[j'-1]$). That means by the end of d steps of the balloon phase all nodes in $[j'-1] \subseteq V$ contain a pebble. But parallel to that balloon phase, the final d iterations of the $(j-1)^{th}$ light phase were also executed. Thus nodes $[j', jg] \subseteq V$ were also pebbled.

For the formal definition of the algorithm we refer the reader to Algorithm 1 and Algorithm 2 in the appendix while further details (including its correctness

7. In our implementations we used a ratio of $R = 3000$ which is given in [BK15] as the Argon2 author's estimate for the case of the hash function used in their design. Regardless, the results in this work are not particularly sensitive the precise value of R as the calls to the hash function represent only a comparatively small proportion of the total cost.

8. For a more formal treatment of these notions and algorithms we recommend looking at [AS15], [AB16].

and complexity) can be found in [AB16].

2.3. Outline of the Results

We describe the hardware constraints of implementing the attack in Section 3. The analysis of the asymptotic quality of the [AB16] attack applied to Argon2i-B can be found in Section 4.

To explore the practicality of this strategy we implemented it in C. That is, for a given σ and τ, the code first samples a fresh Argon2i graph (and builds lists of all parents and children of nodes in the graph). Then it constructs a depth-reducing set $S \subset [n]$ for G and selects appropriate integer g. The precise method for this is described below and depends on whether an additional parallelism bound U is given as input. Next the code simulates an execution of the [AB16] algorithm keeping track of the energy complexity of the execution.[9]

We also implemented several heuristics new to [AB16] for improving the complexity of the executions with the following intuitive goals:

1) Choose the S in a smarter way (Section 5.1).
2) Reduce the cost of a Balloon Phase by maintaining a tighter upper-bound on the number of steps needed to complete each balloon phase (Section 5.2).
3) Reduce the cost of a Light Phase by combining the [BCGS16] attack with [AB16] (Section 5.3).

We analyze the results from the executions in Section 6. In particular the multi-lane variants (where $p > 1$) of Argon2i are discussed in Section 6.8. Finally we implemented and tested the XOR-and-high-indegree counter measure of [CGBS16] to determine its effectiveness at preventing the [AB16] (Appendix 3).

3. Parallelism

In this section we give an example showing how to translate concrete parameters for the [AB16] attack into requirements on the hardware used to implement it. We consider various aspects such as chip size and memory bandwidth. As a reference point, we also give some specifications of consumer grade hardware currently available on the market. We show that the requirements imposed by the [AB16] attack (even for the case of unbounded parallelism) is either already feasible or else will be so in the near future. At the very least we see that upperbounds on parallelism used in some of our more strict experiments, which never-the-less resulted in attacks on practical parameters of the iMHFs, can quite readily be realized with modern semiconductor technology.

3.1. Chip Area and Memory Size

The attacks of [AB16] require parallel computation of the underlying compression function H during the balloon phase. In particular, [AB16] divides the nodes in the DAG into layers, and each layer is divided then into segments of consecutive nodes. Within each layer each segment is re-computed in parallel (the depth-reducing set S eliminates in layer edges so that it is possible to pebble each segment in parallel). Thus, to implement this attack on chip one would need one core (e.g., a Blake2b core) for each segment in a layer. In the theoretical analysis of [AB16] the graph was divided $n^{1/4}$ layers each of size $n^{3/4}$. Each layer in turn was divided into $\sqrt{n}$ segments of $n^{1/4}$ consecutive nodes. Thus, we would need 2^{11} cores to attack $\tau = 4$-pass Argon2i-A with $n = 2^{22}$ nodes ($1KB \times n/\tau = 1GB$ of memory).

The underlying compression function for both versions of Argon2i are based on the Blake2b hash function (though they do differ somewhat from each other). A Blake2b implementation on chip is estimated [BDK16] to take about $0.1mm^2$ of space and DRAM takes about $550mm^2$ per GB. Thus, $5,500$ Blake2b cores would occupy approximately the same amount of space on chip as $1GB$ of DRAM. Thus, we would have space to fit $2^{11} < 5,500$ Blake2b cores needed for the [AB16] attack $\tau = 4$-pass Argon2i with memory $1GB$. However, as parallelism increases so does the required on-chip memory bandwidth (we need to send each core the appropriate values to be hashed during each cycle).

In all of the Argon2i-B instances we evaluated the optimal attack parameters never required more than $1,323$ cores (even without explicitly controlling for parallelism). Thus, space for Blake2b cores does not appear to be a bottle-neck for our attacks. However, as parallelism increases so does the required memory bandwidth. Thus, parallelism would be bounded by the maximum memory bandwidth of our chip.

As we show, it is possible to modify the [AB16] attack to control for the amount of parallelism when constructing the depth-reducing set S.

3.2. Memory Bandwidth

In general we will use bw to denote the bandwidth required to keep a single Blake2b core active. Thus, if

9. We remark that the cost of sampling the graph, set S and g as well as building the parent and children lists is not included in the final complexity. This reflects the fact that such a computation need only be performed a single time and the result can be reused for each subsequent input (e.g. password guess) making the *amortized* contribution of those steps tend quickly towards 0.

we need p cores to instantiate the balloon phase then the chip must have total bandwidth at least $p \times bw$ or else memory bandwidth will become a limiting bottleneck. Implemented on a 4 core machine with 8 threads Argon2i-B uses memory bandwidth $5.8GB/s$ [BDK16, Table 4]. This suggests that we would need memory bandwidth $bw \approx (5.8GB/s)/4 = 1.45GB/s$ for each Blake2b core to keep pace. While this number may vary across different architectures, we will use $1.5GB/s$ as a reference point in our discussion.

At the time that Argon2i-A was developed the maximum bandwidth achieved by modern GPUs was $400GB/s$. Currently, the AMD Radeon R9 Fury graphics cards have a memory bandwidth of 512GB/s [Wal15]. However, recently Samsung has began production of its High Bandwidth Memory 2 (HBM2) chips which will allow for memory bandwidths of well over 1TB/s [Wal16]. Thus, it would be possible to support parallelism up to $p = 666 \approx \frac{1TB/s}{1.5GB/s}$ on a current GPU and even $p = 1000$ in the near future.

In both versions of Argon2i the balloon (and light) phase memory read patterns are pseudo-random but deterministic (i.e. predictable). This potentially allows for significantly leverage prefetching techniques. Furthermore, the memory write pattern is deterministic and has very good locality.[10]

4. Theoretical Analysis of Argon2i-B

Alwen and Blocki [AB16] presented an attack on Argon2i-A, but their paper does not analyze the newest version Argon2i-B — the version from the IRTF proposal [BDKJ16]. In this section we show how to extend the attacks of [AB16] to Argon2i-B. More specifically, Theorem 4.1 achieves attack quality $\Theta\left(N^{0.2}\right)$ by tuning our attack parameters appropriately.

Theorem 4.1. Let Argon2i-B parameters $\tau = O(1)$ and $p = O(1)$ be given and let $N = \tau\sigma$ then there is an attack $\mathcal{A}$ on Argon2i-B with $\mathsf{E\text{-}quality}(\mathcal{A}) = \Theta\left(N^{0.2}\right)$.

PROOF. (sketch) For simplicity we assume $\tau = 1$ and $p = 1$ though the ideas in our analysis easily extend to any constant values τ and p. By [AB16] it suffices to show how to construct a set S of size $|S| = \theta\left(N^{4/5}\right)$ such that $\mathsf{depth}(G - S) = N^{3/5}$. Then we can simply run the algorithm $\mathsf{GenPeb}(G, S, g, d)$ with parameters $g = N^{4/5}$ and $d = N^{3/5}$. GenPeb has complexity $|S|N + gN + dN^2/g = O\left(N^{1.8}\right)$ [AB16] so the attack has quality: $O\left(\frac{N^2}{N^{1.8}}\right) = O\left(N^{0.2}\right)$.

10. In particular balloon phases spend most of their time simply walking along segments in the graph. Only comparatively rarely do they traverse an edge leading to a new layer.

To construct the set S we partition the nodes $1, ..., N$ into equal sized layers $L_1, ..., L_{N^{2/5}}$ each containing $N^{3/5}$ consecutive nodes. For each node $j \leq N$ we add j to the set S if either $j \equiv 0 \mod N^{1/5}$ or if both of j's parents are in the same layer as j. We have $\Pr[\mathsf{parents}(j) \in L_i] \propto \frac{1}{\sqrt{i}}$ thus

$$\begin{aligned} \mathbf{E}\left[|S|\right] &\leq N^{4/5} + \sum_{i=1}^{N^{2/5}} \sum_{j \in L_i} \Pr[\mathsf{parents}(j) \in L_i] \\ &= N^{4/5} + N^{3/5} O\left(\sum_{i=1}^{N^{2/5}} \frac{1}{\sqrt{i}}\right) = O\left(N^{4/5}\right) . \end{aligned}$$

Now any path p in $G - S$ contains at most $N^{1/5}$ nodes from each layer L_i. Thus, $\mathsf{depth}(G - S) \leq N^{3/5}$. □

5. The Implementation

In this section we describe our implementation of [AB16] detailing the various optimization techniques and heuristics we implemented.

5.1. Improved Depth-Reducing Construction for Argon2i Graph

The core of the attacks of [AB16] on Argon2i rely on a construction of a small set S of nodes such that removing S from $G_{\sigma,\tau}$ results in a graph with only short (directed) paths. In a bit more detail, the number $\mathsf{depth}\left(G_{\sigma,\tau} - S\right)$, of edges traversed by the longest path in the remaining graph is small. Before describing our improvements we review the construction of [AB16]. To construct S [AB16] divides the $N = \sigma\tau$ nodes into $\sqrt{d} = N^{1/4}$ layers $L_1, \ldots, L_{\sqrt{d}}$ each containing $N/\sqrt{d} = N^{3/4}$ consecutive nodes. They further divided each layer L_i into $N/d = \sqrt{N}$ segments $L_1^i, \ldots, L_{N/d}^i$ of $\sqrt{d}$ consecutive nodes each. S is now constructed in two steps. First add the last node in every segment L_j^i to S for all $i \leq \sqrt{d}$ and $j \leq N/d$. Then, for each layer L_i and for each node $v \in L_i$ for which $\mathsf{parents}(v) \in L_i$ we add v to the set S. That is we add v if and only if both of v's parents are in the same layer. By removing nodes in S from the graph we ensure that for each layer L_i each of the segments $L_1^i, \ldots, L_{N/d}^i$ in that layer are disconnected from each other . Thus, any path in $G_{\sigma,\tau}$ stays in layer L_i for at most $\sqrt{d} - 1$ step. We have $\mathsf{depth}(G_{\sigma,\tau} - S) \leq d$ as there are $\sqrt{d}$ layers. [AB16] analyzed Argon2i-A and showed that when $d = \sqrt{n}$ the set S will have size $|S| = O(n^{3/4} \ln n)$.

Our first optimization is based on the observation that in Step 2 we do not always need to add v to the

set S even if both of v's parent are in the same layer as v. Instead we only need to add v if these parent edges fail to make progress within a segment. Suppose that v is the a'th node in segment L_j^i and that v's parent u is the b'th node segment $L_{j'}^i$ ($j' < j$). If $b < a$ then we say that the edge (u, v) makes progress in layer L_i. Otherwise if $b \geq a$, we say that the edge (u, v) does not make progress (note that edges $(v-1, v)$ always make progress within a segment so we only need to worry about the pseudorandomly chosen parents). This simple optimization allows us to reduce the size of the set S by a factor of (almost) 2 because (approximately) half of the edges (u, v) will make progress! Furthermore, this optimization does not increase the depth of $G_{\sigma,\tau} - S$. Each edge in p either makes progress within a layer L_i or moves to a higher layer. As before the path p can only make progress $\sqrt{d} - 1$ times within each layer L_i.

Our second optimization is a heuristic one. We use two different parameters gap and $\#layers$ before we divide our N nodes into layers $L_1, \ldots, L_{\#layers}$ of size $N/\#layers$ and we divide each layer into segments $L_1^i, \ldots, L_{N/(\#layers(gap+1))}^i$ of size $gap + 1$ each. We can follow the same construction to find S such that $\mathsf{depth}(G_{\sigma,\tau} - S) \leq gap \times \#layers$. [AB16] fixed $\#layers = (gap + 1) = \sqrt{d}$ to maximize asymptotic performance, but in practice we achieve better attack quality by allowing the two parameters to differ.

5.1.1. Controlling Parallelism. In the GenPeb attack of [AB16] each of we the segments $L_1^i, \ldots, L_{N/(\#layers(gap+1)}$ is re-pebble in parallel. Thus, parallelism $p = N/(\#layers(gap + 1))$ is sufficient to execute our attack. If we have an upper bound U on parallelism then we can select the parameters $\#layers$ and gap subject to the condition that $(gap + 1)\#layers \geq N/p$.

5.2. Dynamic Reduction of Balloon Phase Costs

Recall that the balloon phase is used in [AB16] to recover pebbles that were discarded during the last light phase. Balloon phases, unlike light phases, can be memory intensive. Therefore, to minimize cumulative memory usage it is imperative to minimize the running time of the balloon phase. In [AB16] each balloon phase is (pessimistically) assumed to run for exactly d steps, where $d \geq \mathsf{depth}(G_{\sigma,\tau} - S)$. We use a simple observation to reduce the cost incurred during balloon phases. The observation is that most balloon phases never need to run for the full d steps to recover the necessary pebbles for the next light round. If we begin a balloon phase on round i, where node i is in layer L_j with $j = \lceil \frac{i \times \#layers}{N} \rceil$, then we only need to recover pebbles on nodes in layers $L_1, \ldots, L_j$ during the balloon phase. If we remove nodes in the set S and layers $L_{>j} = \bigcup_{k=i+1}^{N/\#layers} L_k$ from the graph then we have $\mathsf{depth}\big(G_{\sigma,\tau} - S - L_{>j}\big) \leq j \times gap$. Thus, the balloon phase will only need to run for $j \times gap$ steps. On average a balloon phase will only needs to run for about $d/2$ steps.

5.3. Incorporating Memory-Reducing Attack

Our second observation is that the attacks of [AB16] can be combined with the opportunistic memory-reducing attacks of [BCGS16]. The attack of [BCGS16] was based on the simple observation that you could discard the pebble on node v as soon as we finish pebbling the last of v's children. While this attack only reduces memory consumption by a constant factor (in contrast to the asymptotic reductions in [AB16]), the constant factors were large enough that Argon2i was updated in response. In Argon2i-B each new block that is being stored in memory is first XORed with the existing block in memory that is being replaced. In the language of graph theory this means that each node $v > \sigma$ has an additional parent $v - \sigma$, where σ is the size of the memory window. This modification ensures that in the attack of [BCGS16] we cannot discard a pebble early (e.g., for each node v we will not finish pebbling v's children until the exact moment that v is outside the memory window).

However, if we are running the attack of [AB16] the current memory window will be 'interrupted' by a balloon phase. We can potentially discard the pebble on v well before we have pebbled the last of v's children if we know that we have an opportunity to recover v before it is needed again. In particular, if we know that we can recover a pebble on node v during the next balloon phase we can discard the pebble on node v as soon as it is no longer needed for the current light phase or as soon as we finish pebbling the last of v's children that we need to pebble in the current light phase. More formally, if the light phase starts at time t and ends at time $t + g - 1$ then we can discard a pebble from node $v \notin S$ during round $t' < t + g - 1$ if $\forall u \in [t', t + g - 1]$ we have $v \notin \mathsf{parents}(u)$.

5.4. Attack Implementation

We developed C code to simulate the [AB16] attack on randomly generated Argon2i-A, Argon2i-B and BH instances. Our code is available at GitHub repository https://github.com/ArgonAttack/AttackSimulation.git. Our implementation includes the additional optimizations described in this section. Specifically, our code base includes the following procedures:

1) `GenerateRandom(__)DAG`. Procedures to sample a random Argon2i-A, Argon2i-B or BH DAG $G_{\sigma,\tau}$ given memory parameter σ and the parameter τ specifying the number of passes over memory.
2) `SelectSetsS`. A procedure to select the depth reducing set S for an input DAG $G_{\sigma,\tau}$ given input parameters gap and $\#layers$.
3) `Attack`. Procedures to simulate the optimized [AB16] attack. Specifically, `Attack` simulates one iteration at a time and measure cumulative energy costs (memory usage + calls to hash function). `Attack` takes as input the DAG $G_{\sigma,\tau}$, the depth reducing set S along with the associated parameters gap and $\#layers$ and a parameter g which specifies the length of each light phase. The procedure returns an (upper bound[11]) on the cost of the optimized [AB16] attack.
4) `SearchForg`. A procedure to search for the best g value. The procedure takes as input a DAG G, the depth reducing set S along with the associated parameters gap and $\#layers$. The procedure then uses an iteratively refining grid search heuristic to search for the optimal parameter g to use in the attack. In more detail, we start with a large range $[gMin, gMax]$ of potential g values containing the point $g = n^{3/4}$ (the value of g used in the theoretical attacks of [AB16])[12]. In the first iteration we repeatedly run `Attack` to measure attack costs when we instantiate g with each value in the set $\{gMin, gMin + gStep, \ldots, gMin + 8 \cdot gStep, gMax\}$ where $gStep = \frac{gMax-gMin}{9}$. Suppose that the value $g = gMin + i \cdot gStep$ yielded the lowest attack cost. Then in the next iteration we would set $gMin = \max\{gMin, gMin + (i-1)gStep\}$ and we would set $gMax = \min\{gMax, (i+1)gStep\}$. We repeat this process 6 times in total and return the best value of g that we found.
5) `SearchForSParameters`. A procedure to search for the best pair of parameters gap and $\#layers$ which control the construction of the set S. The procedure takes as input the DAG G and a parameter g. The procedure `SearchForSParameters` is similar to `SearchForg` except that the iteratively refining grid search is carried out in two dimensions. For each pair of parameters $(gap, \#layers)$ we must run `SelectSetsS`$(gap, \#layers)$ to construct the depth-reducing set S before we can call `Attack` to simulate the attack.
6) `SearchForSParametersWithParallelism`. Similar to `SearchForSParameters` except that the procedure additionally takes as input a parallelism parameter. The parameters gap and $\#layers$ are chosen subject to the constraint that $p \approx N/(\#layers(gap+1))$ so that the attack uses parallelism p. The iteratively refining grid search is carried out in one dimension[13].

5.4.1. Advantages of Simulation Over Theoretical Analysis. By simulating the [AB16] attack we need not rely on pessimistic assumptions to upper bound attack costs. For example, in the theoretical analysis of [AB16] they assume that the pebbling algorithm has to pay to keep pebbles on every node in the depth-reducing set S during every pebbling round (total cost: $n|S|$). While this assumption may be necessary for a theoretical analysis (the set S is only defined once we specify a specific instance G), it overestimates costs paid during many pebbling rounds (especially during early pebbling rounds when we will have very few pebbles on the set S).

6. Analysis & Implications

In the simulation of [AB16], after fixing iMHF parameters τ and $n = \sigma\tau$ we first generated a random instances of the Argon2i-A (resp. Argon2i-B, BH) DAG $G_{\sigma,\tau}$. We then temporarily set $g = n^{3/4}$ (the value of g used in the theoretical analysis from [AB16]) and used the procedure `SearchForSParameters` (2-dimensional iteratively refined grid search) to find good values for the attack parameters $\#layers$ and gap. Once we have $\#layers$ and gap we then ran `SearchForg` (1-dimensional iteratively refined grid search) to find a good value for the attack parameter g. Once we have all of the attack parameters $g, gap, \#layers$ we sampled 9 additional random Argon2i-A DAGs (resp. Argon2i-B,

11. While the real-world attack would be carried out on a massively parallel machine the simulations were carried out on a single-threaded process. To make the `Attack` simulation as efficient as possible we allowed the `Attack` procedure to overestimate the cost of the attack in order to improve efficiency. In particular, the procedure could occasionally double count the number of pebbles on (up to) $\mathsf{depth}(G_{\sigma,\tau} - S)$ parent nodes during a balloon phase. These double counted costs comprise a very small fraction of the total energy cost. Thus, we may overestimate the cost of the attack, but only very slightly. In any case double counting these few pebbles can only cause us to underestimate attack quality.

12. We also require that $gMin \geq gap \times \#layers$ so that we ensure that the depth of the graph does not exceed the length of a balloon phase.

13. Once we fix one of the parameters (e.g., $\#layers$) the other parameter (e.g., gap) is fully specified.

BH), ran `SelectSetsS` to generated a depth reducing set S for each DAG and measured using our simulation algorithm `Attack`. In addition to recording attack costs for each iMHF instance we also recorded the optimal attack parameters, required parallelism and the size of the depth reducing sets constructed.

6.1. Memory Consumption and Runtime

Recall that memory parameter σ denotes the table size used by the honest algorithm and τ denotes the number of passes over that table. Then $n = \sigma\tau$ is (roughly) the number of calls the honest algorithm makes to the hash function and so is a reasonable approximation of the runtime of the honest algorithm.

For the Argon2i-B and BH iMHFs we simulated our attack for each of pair of parameters $n \in \{2^{17}, 2^{18}, 2^{19}, 2^{20}, 2^{21}, 2^{22}, 2^{23}, 2^{24}\}$ and $\tau \in \{1, 3, 4, 6, 10, 20\}$. We denote the label size (in bytes) by B. For example the Black2b based hash function of Argon2 has an output of size $B = 1024$ while BH we use $B = 512$ as this is the output of all considered hash functions in that work (with the exception of SHA3 where a $B = 1344$ is used). Note that the actual memory usage (by the honest algorithm) is then $M = B\sigma = Bn/\tau$. Thus for Argon2i-B, when $\tau = 1$, then $n = 2^{24}$ nodes corresponds to $M = 16GB$ of memory usage, but when $\tau = 10$, $n = 2^{24}$ nodes corresponds to $M = 1.6GB$ of memory usage. We also simulated our attack on τ-pass Argon2i-A for $\tau \in \{1, 3, 4\}$ to compare Argon2i-A and Argon2i-B.

6.2. Argon2i-B

Figure 1 shows attack quality against Argon2i-B for different memory parameters σ.[14]. The plots demonstrates that the attacks are effective even for "pessimistic" parameter settings (e.g., at $\tau = 6$ passes over $1GB$ of memory the attack already reduces costs by a factor of 2). To prevent any attack at $1GB$ of memory we would need to select $\tau > 10$. While attack quality decreases with τ, we stress that it is undesirable to pick larger values of τ in practice because it increases running time by a factor of τ. Human users are not known for their patience during authentication. If we select a larger value of τ then we must select σ small enough so that we can make τ passes over memory before the user notices or complains (e.g., after 1–2 seconds).

14. Each data point measures the average attack quality over 10 random samples. We do not include error bars because attack cost was showed minimal variation across all 10 samples in all iMHF instance that we tried. For example, the energy cost of our attack on all 10 graphs matched on first 2-3 significant digits.

The IRTF proposal [BDKJ16] selects τ as follows: First determine the maximum memory usage M that each instance of Argon2i-B can afford (setting $\sigma = M/B$) as well as the maximum allowable running time t that each call can afford. Then select the maximum τ such that we can complete τ passes through M memory in time t. This procedure could easily result in the selection of the parameter $\tau = 1$ pass through memory (e.g., in settings where lots of memory is available and/or users are less patient). In this case attack quality is approximately 5 at 1GB of memory and approximately 9.3 at 16GB (the latter data point is outside the visible range of Figure 1).

At first glance our analysis may seem to suggest that one should select τ large since $\mathsf{E\text{-}quality}(\mathcal{A})$ increases as τ decreases. However, we stress that maximizing total cost $\mathsf{ec}_R(\mathcal{A})$ for an attacker is not necessarily equivalent to minimizing attack quality $\mathsf{E\text{-}quality}(\mathcal{A})$. In particular, $\mathsf{E\text{-}quality}(\mathcal{A})$ simply measures the ratio between the cost of our attack and the cost of the naive evaluation algorithm where the latter cost increases as τ decreases. When memory is not a limited resources it may be reasonable to simply set τ to maximize the cost of the attack $\mathsf{ec}_R(\mathcal{A}) = \mathsf{ec}_R(\mathcal{N})/\mathsf{E\text{-}quality}(\mathcal{A})$ subject to the constraint that $\tau\sigma = n$ (e.g., in a setting where total running time is limited by user patience, but memory is not a limiting factor). We have

$$\mathsf{ec}_R(\mathcal{A}) \approx n^2 \frac{\left(\frac{1}{\tau} - \frac{1}{2\tau^2}\right)}{\mathsf{E\text{-}quality}(\mathcal{A})}$$

so that it is not necessarily optimal to maximize τ.

Figure 2 plots $\mathsf{ec}_R(\mathcal{A})/n$ as τ varies for $n \in \{2^{18}, 2^{20}, 2^{22}\}$. When $n = 2^{16}$ we achieve the maximum value of $\mathsf{ec}_R(\mathcal{A})$ when $\tau = 4$ and when $n = 2^{22}$ the maximum value is achieved when $\tau = 10$.

6.3. Argon2i-A vs. Argon2i-B

Figure 3 compares attack quality against both versions of Argon2i. On a positive note the results show that the updated version of Argon2i is indeed somewhat less susceptible to the attacks of [AB16]. On a negative note the attacks on Argon2i-A, the version from the password hashing competition, are quite strong. For example, even at $\tau = 4$ passes through memory (which increases running time by a factor of 4) we get an attack with quality > 4. Meaning that the adversary can reduce his energy costs by a factor of 4.

Comparison with [AB16]. To make an explicit comparison with [AB16] consider Argon2i-A with $\tau = 3$-passes and look at [AB16, Figure 1] (similar comparisons apply for 1,4,7,10 passes). [AB16] achieves $\mathsf{E\text{-}quality}(\mathcal{A}) = 2$ around $n = 2^{23}$ (equivalently,

$\sigma = 2^{23}/3$ memory blocks or 2.8GB). By comparison, we achieve $\mathsf{E}\text{-quality}(\mathcal{A}) = 2$ against Argon2i-A for $n \approx 3 \times 2^{16}$ (equivalently,$\sigma = 2^{16}$ memory blocks or 66MB). We believe this to be a significant reduction in the memory size for which attack $\mathsf{E}\text{-quality}(\mathcal{A}) \geq 2$! Arguably, in practice one would very rarely use >2.8GB of memory for password hashing, but one would always want to use at least 66MB. As we previously noted [AB16] does not analyze Argon2i-B so we cannot draw explicit comparisons. However, we note that our attack on Argon2i-B still achieves attack quality ≥ 2 around $n \approx 3 \times 2^{18}$ (equivalently,$\sigma = 2^{18}$ or 262MB). Thus, our attack $\mathcal{A}$ on the improved Argon2i-B achieves higher attack quality $\mathsf{E}\text{-quality}(\mathcal{A})$ than the [AB16] attack on Argon2i-A.

6.4. Balloon Hashing

Figure 4 shows attack quality against the BH iMHF scheme of [BCGS16] as memory usage M varies. We remark that the design of BH is flexible and that it can be instantiated with many different hash functions. Consequently, the block size may vary depending on the instantiation. As mentioned above we use label size of $B = 512B$ to generate our plots (i.e., instantiated with Blake2b)[15] and that we assume that the indegree parameter $\delta = 3$. Thus, when $\tau = 1$, setting $n = 2^{24}$ corresponds to $M = 8GB$ of memory usage in comparison to $M = 16GB$ for Argon2i-B with the same parameters. The attacks of [AB16] perform particularly well when the block size is small. In particular, the (potentially) smaller blocksize of BH is a disadvantage (in comparison with Argon2i-B) as it means that we need to select a higher value of σ to achieve the same memory usage and the attack quality of [AB16] increase rapidly with n. This would make BH less resistant to the attacks than Argon2i-B.

6.5. Attack Parameters

Table 1 shows how the parameters of our attack (e.g., $g, \#layers, gap$ parallelism p) vary with memory usage M and τ. A few interesting trends emerge. First, the maximum level of parallelism needed for any of the Argon2i-B instances that we tried was $1,324$ ($\tau = 1$-pass Argon2i-B with $M = 16GB$) and for Argon2i-A parallelism never exceeded $1,496$ ($\tau = 1$-pass Argon2i-A with $M = 16GB$). In Section 6.6 we explore how attack quality is affected if we explicitly upper bound parallelism. Second, the amount of parallelism p needed tends to increase with σ, but decreases as we

15. If BH was instantiated with a hash function that works over $B = 512$ bit block sizes then the graphs would be shifted left three places. Similarly, if BH was instantiated with block size $B = 1KB$ then the graph would be right shifted one place.

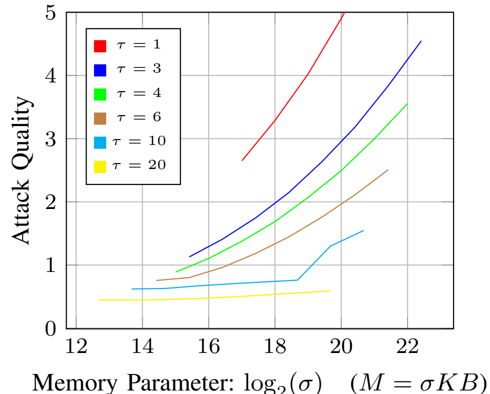

Figure 1: Argon2i-B Attack Quality

increase the number of passes through memory τ. For example, $\tau = 4$-pass Argon2i-B at $M = 1GB$ only uses parallelism 400. Interestingly, the size $|S|$ of the depth reducing set did not seem to vary much as τ increases (holding $n = \sigma\tau$ constant)[16]. Interestingly, the procedure `SearchForSParameters` tends to select the parameters $\#layers$ and gap so that $\#layers$ is several times larger than gap (in the theoretical analysis of [AB16] these parameters were equated). As expected the attack parameters gap, L and g all seem to increase with $n = \sigma\tau$, the number of nodes in the DAG $G_{\sigma,\tau}$. The Bandwidth-On-Chip column estimates the amount of memory bandwidth on chip necessary to support p cores (under the assumption that we need $1.5GB/s$ per core). In the worst case ($M = 1GB$ and $\tau = 1$ memory passes) we need on-chip bandwidth of about 2 TB/s, a value that may be plausibly achieved in the near future (there is currently a chip that achieves memory bandwidth of 1 TB/s). In most other instances the required bandwidth is significantly reduced (e.g., $0.6TB/s$ for 4-pass Argon2i-B at $1GB$).

6.6. Controlling Parallelism

We now explore how attack quality is affected when parallelism is limited (e.g., due to limitations on on-chip bandwidth). We run two experiments. The first involves Argon2i-A and Argon2i-B and the second involves Argon2i-A. In the first experiment we

16. $|S|$ does seem increase slightly with τ, but this trend is hidden because Table 1 only reports the two most significant digits of $|S|$. Furthermore, we observed the opposite trend ($|S|$ decreases slightly with τ) for the BH iMHF. It is also worth noting that for each iMHF instance (τ, σ) that we tested we sampled 10 distinct graphs $G_{\sigma,\tau}$ and we had to sample different sets S for each graph. We found that the size of the $|S|$ never varied greatly across these 10 different random instances. In fact, if we only consider the two most significant digits then the size of $|S|$ was always the same.

iMHF	N	τ	M	g	$\#layers$	gap	p	$\|S\|$	Bandwidth-On-Chip (TB/s)
Argon2i-B	24	1	$16GB$	$381,376$	264	47	$1,324$	$1.0e6$	1.986 TB/s
Argon2i-B	22	4	$1GB$	$93,237$	228	45	400	$3.1e5$	0.6 TB/s
Argon2i-B	24	4	$4GB$	$220,808$	388	47	901	$1.0e6$	1.352 TB/s
Argon2i-B	24	6	$2.7GB$	$223,702$	512	47	683	$1.0e6$	1.025 TB/s
Argon2i-B	24	10	$1.6GB$	$218,137$	512	78	415	$1.0e6$	0.6225 TB/s
Argon2i-A	20	4	$256MB$	$65,555$	113	36	251	$4.7e4$	0.376 TB/s
Argon2i-A	22	4	$1GB$	$155,394$	156	47	561	$1.4e5$	0.8415 TB/s
Argon2i-A	24	4	$4GB$	$357,096$	233	75	948	$3.8e5$	1.422 TB/s
BH	24	1	$8GB$	$357,096$	170	65	$1,496$	$4.2e5$	2.244 TB/s
BH	24	4	$2GB$	$215,223$	233	75	948	$3.8e5$	1.422 TB/s
BH	24	10	$0.8GB$	$192,915$	388	78	548	$3.7e5$	0.822 TB/s

TABLE 1: Best Attack Parameters Found (Selected Argon2i-A,B and BHLin Instances).

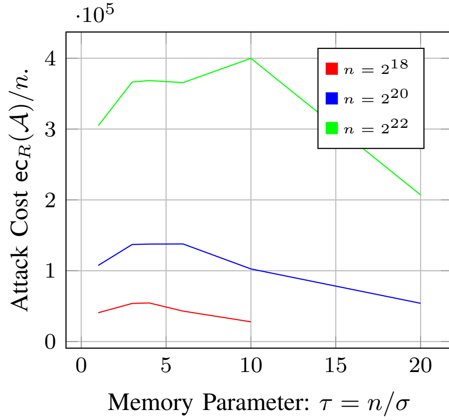

Figure 2: Argon2i-B Attack Cost

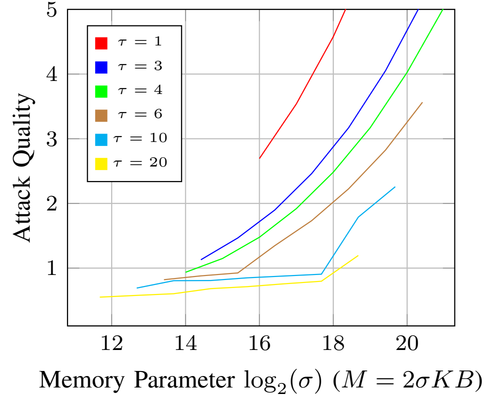

Figure 4: Balloon Hash Attack Quality

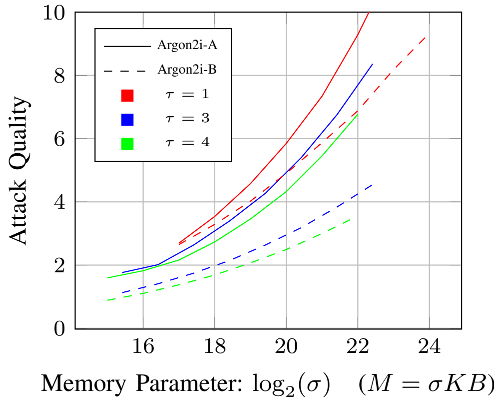

Figure 3: Argon2i-A vs. B.

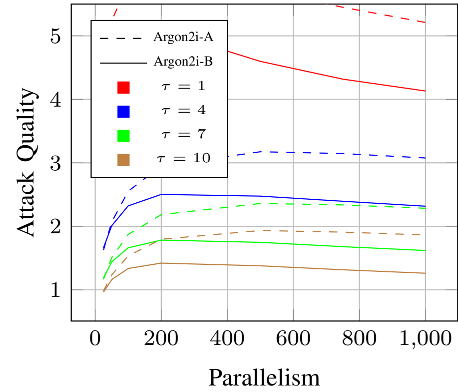

Figure 5: Argon2i Attacks with Bounded Parallelism $(M = 1GB)$

fix memory $M = 1GB$ $(\sigma = 2^{20})$ and select $p \in \{25, 50, 100, 200, 500, 750, 1000\}$ and $\tau \in \{1, 4, 7, 10\}$ and generate a random Argon2i-B (resp. Argon2i-A) instance $G_{\sigma,\tau}$. Once again fixing $g = n^{3/4}$ we use the procedure `SearchForSParametersWithParallelism` to find good attack parameters gap and $\#layers$, subject to the condition that $(gap+1)\#layers = n/p$ so that the attack uses parallelism *exactly* p. We then use the procedure `SearchForg` to find a good parameter g. Finally, we generate 10 instances of graphs $G_{\sigma,\tau}$, run `SelectSetS` to generate the depth-reducing set S for each instance and run `Attack` to find the cost of each attack. Figure 5 shows the results of this experiment.

The second experiment is similar except that we use Argon2i-A and we fix runtime $n = 2^{24}$ instead of memory. We select $p \in \{25, 50, 100, 200, 500, 1000\}$

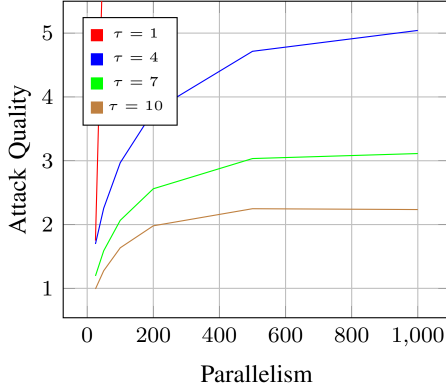

Figure 6: Argon2i-A $(n = 2^{24}, M = n/\tau KB)$

and $\tau \in \{1, 4, 7, 10\}$ and generate a random Argon2i-B instance $G_{\sigma=n/\tau,\tau}$. We use the same procedures `SearchForSParametersWithParallelism` and `SearchForg` to find our attack parameters $g, gap, \#layers$ before simulating our attack. Figure 6 shows the results of this second experiment.

6.7. Discussion

In Figure 6 attack quality (almost) monotonically increases with parallelism (excluding the plot $\tau = 10$-pass which peaks at $p = 500$). Notice that attack quality increases rapidly with p when p is small, but this rate of increase slows dramatically as p increases. In Figure 5 attack quality for Argon2i-B peaks at around $p = 200$ and often starts to *decrease* as parallelism increases after this point. It may seem surprising that attack quality decreases with parallelism, but remember the procedure `SearchForSParametersWithParallelism` ensures that we construct the set S in such a way that we need parallelism exactly p. Thus, the plots are suggesting that the optimal attack will not use parallelism $p > 200$ (whether or not we control for parallelism). In this case the attack could be implemented on a chip with only $300GB/s$ of memory bandwidth (using the same assumption that we need $1.5GB/s$ of bandwidth per core).

6.8. Multiple Lanes

Thus, far in our analysis of Argon2i-A and Argon2i-B we have focused on the single-threaded version of the iMHFs. In this section we discuss to possible ways that an iMHF could be extended to support parallelism: a trivial extension and the more-detailed approach taken by Argon2i. Surprisingly, we find that the trivial extension offers better resistance to the [AB16] attacks.

6.8.1. Trivial Extension. Given a single-threaded iMHF the easiest way to support parallelism $p > 1$ would have been evaluate p independent instances of the iMHF in parallel and then hash the final block from each iMHF instance. More specifically, given parameters τ, σ and p each individual iMHF instance would have memory parameter σ/p and make τ passes over memory. This solution does give the adversary an easy time-memory trade-off. Namely, an adversary with σ/p memory could still evaluate the iMHF, but it would also increase his running time by a factor of p so this attack does not reduce overall energy costs or AT complexity. Because energy complexity scales linearly with the number of instances being computed, attack quality against the multi-threaded iMHF with parameters τ, σ and p will be equal to the attack quality on the single threaded variant of the iMHF with parameters $\tau, \sigma/p$. Thus, increasing parallelism p will increase resistance to the [AB16] attacks because their attack quality grows with σ.

6.8.2. Argon2i Approach. In an attempt to avoid this time-memory trade-off the Argon2i designers took a different approach. They divide memory into p lanes each with space for σ/p node labels and each lane is further divided into 4 slices. Each thread will be responsible for filling in one lane, but the value of a node in one lane is allowed to depend on the values of nodes in other lanes. In particular, to pick the parent of a node v we first select a lane uniformly at random, and then we pick a random parent from that lane according to a non-uniform distribution that is specified in the IRTF proposal [BDKJ16] (the exact specification of the distribution is not important here). To prevent blocking they further require that the i'th node in a lane cannot be dependent on parent node from another slice if the parent node is in the same 'slice' of memory, where each memory slice contains $\sigma/(4p)$ nodes in each lane. Loosely, this means that if i's parent is the j'th node then $i - j$ must be somewhat large (about $\sigma/(4p)$).

6.8.3. Analysis and Discussion. We observe that the approach taken in the design of Argon2i can actually *decreases* resistance to the [AB16] attack when compared with the trivial approach to supporting parallelism. Recall that during the construction of the depth-reducing set S we do not need to add a node v to the set S if its parent is in the same layer. However, the parent of node v will come from a different lane with probability $(p-1)/p$ and whenever we select v's parent from a different lane we are almost guaranteed that v's parent will *not* be in the same layer because the node must occur in an earlier memory slice.[17]

As a concrete example we set $\tau = 4$, $\sigma = 2^{17}$ and $p = 4$ ($n = 2^{19} = \tau\sigma$) and generated random Argon2i-

17. Typically, we will have $n/\#layers < \sigma/(4p)$ (the size of an individual layer).

B DAGs. We achieve attack quality > 1.18 even with minimal effort to optimize the parameters used in the attack [18]. By comparison, if Argon2i-B had followed the trivial approach to support parallelism then [AB16] would yield no attack against Argon2i-B with these particular parameters (even with our optimizations). Specifically, we would have had 4 independent 4-pass Argon2i-B instances with memory parameter $\sigma = 2^{17}$. The best attack quality we found in the previous section on 4-pass Argon2i-B with $\sigma = 2^{17}$ was $0.89 < 1$. We conjecture that attack quality on multi-lane versions of Argon2i-A and Argon2i-B can be further improved with additional effort to optimize the parameters used in the attack and the heuristics used to construct the depth-reducing set S. We leave this an interesting challenge for future research. In this paper we have chosen to focus on the single-lane variations of Argon2i-A and Argon2i-B since, combined with the trivial parallelism approach, they lead to iMHFs that are *more* resistant to the Alwen-Blocki attack.

7. Conclusions

We proved that the Alwen-Blocki attack [AB16] on Argon2i-A can be extended to Argon2i-B, and we provided several novel techniques to improve this attack. It was previously believed that the Alwen-Blocki attack [AB16], while it does yield large asymptotic reductions in energy cost as σ grows large, was not relevant for practical parameter ranges (e.g., $\leq 16GB$ of memory). We use simulations to show that, with our optimizations, the Alwen-Blocki attack [AB16] is already relevant for practical parameter ranges. In fact, even for 'pessimistic' parameter settings ($\tau = 6$) the attack can reduce costs by a factor of 2 when using just $1GB$ of memory. We also showed that when on-chip memory bandwidth limits parallelism we can adjust attack parameters accordingly without significantly decreasing attack quality. However, our results show that Argon2i-B offers better resistance to the attack than Argon2i-A and than the balloon hashing algorithm. Ultimately, there is a need for continued cryptanalysis of iMHFs such as Argon2i-A and Argon2i-B as there are many other attack heuristics which could potentially yield additional cost reductions for the attacker in practice. Could our attack on Argon2i-B be improved in the future? Can we lower bound the energy complexity necessary to evaluate Argon2i-B?

In contrast to general cryptanalysis, in password cracking for real world password distributions a small reduction in cost can significantly increase the fraction of cracked passwords. A rational adversary stops attacking when marginal guessing costs (i.e., cost to evaluate iMHF) exceed marginal benefits (e.g., the value of the cracked password times the probability that the next password guess is correct). Reducing iMHF costs (e.g., by a factor of 10) can greatly increase % of compromised passwords in an offline attack (e.g., by up to 60% [BD16, Figure 1]). It remains an interesting direction for future research to understand the concrete financial costs involved in implementing iMHF attacks such as the Alwen-Blocki attack [AB16] in hardware.

18. We found the attack parameters $gap = 32$, $\#layers = 256$ and $g = 13,970$ by hand.

References

[AB16] Joël Alwen and Jeremiah Blocki. Efficiently Computing Data-Independent Memory-Hard Functions. In *Advances in Cryptology CRYPTO'16*, pages 241–271. Springer, 2016.

[AS15] Joël Alwen and Vladimir Serbinenko. High Parallel Complexity Graphs and Memory-Hard Functions. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '15, 2015. http://eprint.iacr.org/2014/238.

[Aum16] JP Aummason. What's up argon2? BSidesLV 2016, 2016. Slides Available at https://speakerdeck.com/veorq/whats-up-argon2.

[BCGS16] Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. Balloon hashing: Provably space-hard hash functions with data-independent access patterns. Cryptology ePrint Archive, Report 2016/027, Version: 20160601:225540, 2016. http://eprint.iacr.org/.

[BD16] Jeremiah Blocki and Anupam Datta. CASH: A cost asymmetric secure hash algorithm for optimal password protection. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 371–386, 2016. http://arxiv.org/abs/1509.00239.

[BDK15] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Fast and tradeoff-resilient memory-hard functions for cryptocurrencies and password hashing. Cryptology ePrint Archive, Report 2015/430, 2015. http://eprint.iacr.org/2015/430.

[BDK16] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2 password hash. Version 1.3, 2016. https://www.cryptolux.org/images/0/0d/Argon2.pdf.

[BDKJ16] Alex Biryukov, Daniel Dinu, Dmitry Khovratovich, and Simon Josefsson. The memory-hard Argon2 password hash and proof-of-work function. Internet-Draft draft-irtf-cfrg-argon2-00, Internet Engineering Task Force, March 2016.

[Ber] Daniel J. Bernstein. Cache-Timing Attacks on AES.

[BK15] Alex Biryukov and Dmitry Khovratovich. Tradeoff cryptanalysis of memory-hard functions. Cryptology ePrint Archive, Report 2015/227, 2015. http://eprint.iacr.org/.

[CGBS16] Henry Corrigan-Gibbs, Dan Boneh, and Stuart Schechter. Balloon hashing: Provably space-hard hash functions with data-independent access patterns. Cryptology ePrint Archive, Report 2016/027, Version: 20160114:175127, 2016. http://eprint.iacr.org/.

[DKW11] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 125–143. Springer, 2011.

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

[HP70] Carl E. Hewitt and Michael S. Paterson. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation. chapter Comparative Schematology, pages 119–127. ACM, New York, NY, USA, 1970.

[Kho16] Dmitry Khovratovich. Re: [Cfrg] Balloon-Hashing or Argon2i. CFRG Mailinglist, June 2016. https://www.ietf.org/mail-archive/web/cfrg/current/msg08282.html.

[Per09] C. Percival. Stronger key derivation via sequential memory-hard functions. In *BSDCan 2009*, 2009.

[PHC] Password hashing competition. https://password-hashing.net/.

[Tho79] Clark D. Thompson. Area-time complexity for VLSI. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 81–88. ACM, 1979.

[Wal15] Mark Walton. Ars Technica Article: The R9 Fury is AMD'ss best card in years, but just who is it for? http://bit.ly/2aYrZ2j (Retrieved 8/5/2016), 2015.

[Wal16] Mark Walton. Ars Technica Article: Graphics cards with 1024GB/s bandwidth? Samsung begins HBM2 production. http://bit.ly/2b0ryRE (Retrieved 8/5/2016), 2016.

Appendix

The [AB16] Algorithm

The following figures are taken almost verbatim from [AB16] (after appropriate notational changes) and gives the high level pseudo-code for the attack described in that work and implemented in this one.

Here need and keep are defined in Algorithm 2 and Algorithm 3 respectively. The notation L_i denotes the ith layer of nodes as in Section 5.1 and these layers are further divided into segments of $gap+1$ consecutive nodes. Intuitively, the function need specifies that during a balloon phase we will fill in each of these gaps in parallel one node at a time moving on to layer L_{i+1} as soon as we completely finish repebbling layer L_i. The function keep essentially allows us to discard pebbles during a balloon phase as soon as they pass outside the current memory window (e.g. every node in L_{i+1} is at least σ nodes ahead of v then we can discard a node v as soon as we finish re-pebbling layer L_i).

Algorithm 1: GenPeb (G, S, g, d)

Arguments : $G=(V,E)$, $S \subseteq V$, $g \in [\mathsf{depth}(G-S), |V|]$, $d \geq \mathsf{depth}(G-S)$
Local Variables: $n = |V|$

```
1  for i = 1 to n do
2      Pebble node i.
3      l ← ⌊i/g⌋ * g + d + 1
4      if i mod g ∈ [d] then            // Balloon Phase
5          d' ← d − (i mod g) + 1
6          N ← need(l, l + g, d')
7          Pebble every v ∈ N which has all parents pebbled.
8          Remove pebble from any v ∉ K where K ← S ∪ keep(i, i + g) ∪ {n}.
9      else                             // Light Phase
10         K ← S ∪ parents(i, i + g) ∪ {n}
11         Remove pebbles from all v ∉ K.
12     end
13 end
```

Algorithm 2: Function: $\mathsf{need}(x, y, d')$

Arguments: x, $y \geq x$, $d' \geq 0$
Constants : Pebbling round i, g, gap.

1 $j \leftarrow (i \mod g)$ `// Current Layer is` $L_{\lfloor j/gap \rfloor}$
2 **Return** $L_{\lfloor j/gap \rfloor} \cap \left\{ i \cdot gap + j \;\middle|\; i \leq \frac{n}{gap} \right\}$

On the XOR-and-High-Indegree Trick

At a high level the current Balloon Hashing DAG [BCGS16] has similar structure to the Argon2i iMHFs (the underlying hash functions are different). Both DAGs have indeg $=3$. However, the original version of the Balloon Hashing algorithm (BHLin) [CGBS16] had indeg $=21$. Besides $v-1$ a node v had 20 other parents chosen uniformly at random so that the label of node v depends on up to 21 different labels. Of course, applying an iterative Merkle-Damgard

Algorithm 3: Function: $\mathsf{keep}(x, y)$

Arguments: x, $y \geq x$
Constants : Pebbling round i, g, gap, $\#layers$, n, σ.

1 $j \leftarrow (i \mod g)$
2 $\ell \leftarrow \lfloor (j/gap \rfloor$ `// Current Layer`
3 **Return** $L_{\geq \ell - \lceil \frac{\sigma \#layers}{n} \rceil}$

construction to hash these 21 labels would result in a dramatic slowdown[19] BH-DoubleBuffer [CGBS16] avoided Merkle-Damgard by applying a cheap linear operator (e.g., XOR) to the 21 labels before we apply the underlying hash function.

It seems like this trick could potentially make attacks of Alwen and Blocki [AB16] much less efficient in practice. In particular, the attack keeps pebbles on all of the parents of the next g nodes that we want to pebble before the end of the light phase. However, there are up to $20g$ parents so the memory costs could be quite high in practice. The increased indegree will also increase the probability that a node v needs to be included in the depth-reducing set (v needs to be included if *any* of the edges from its parents fail to make progress in its layers). Does this XOR trick increase resistance to the attacks of Alwen and Blocki [AB16]?

To address this question we introduce a new hypothetical iMHF called iXOR. iXOR is Argon2i-A with the modification that we pick 20 random parents for each node v in addition to $v-1$ and XOR the labels together before applying the underlying hash function. iXOR is similar in spirit to the old balloon hashing algorithm BH-DoubleBuffer [CGBS16]. We evaluate our attack on iXOR so that we can isolate the effect of the XOR trick on attack quality. Our goal is in this section is to evaluate the effect of the XOR trick on attack quality as we evaluate attack quality on the current balloon hashing algorithm paper.

Figure 7 plots attack quality vs. memory against the iXOR construction when instantiated with the same hash function used in Argon2. The dotted lines plot attack quality when we implement the attacks of Alwen and Blocki [AB16] along with the other optimizations described earlier in the paper. These plots seem to indicate demonstrate that the XOR hash trick dramatically increases attack quality in practice. However, we introduce an additional optimization called XOR hash which dramatically improves attack quality.

1. XOR Compression

We can dramatically improve attack quality by using a trick we call XOR hash. The basic observation is that instead of storing the labels for up to $20g$ parents we observe that we can compress these labels at the end of the balloon phase. For each node v that we want to pebble in the next light phase we do not need to store the labels of all of v's parents we can just store the XOR of these labels[20]. While this optimization doesn't change asymptotic performance it significantly improves attack quality for practical parameter ranges.

2. Discussion

For the purposes of comparison, the green line in Figure 7 shows attack quality against Argon2ib. The plot shows that attack quality against 3-pass iXOR and 6-pass Argon2ib are roughly equivalent. The 3-pass iXOR DAG has $3 \cdot 2^m$ nodes while the 6-pass Argon2ib nodes has $6 \cdot 2^m$ nodes. Thus, the XOR trick from [CGBS16] potentially has some benefit. If it takes the same amount of time to label nodes in iXOR and Argon2ib then 3-pass iXOR would be preferable to 6-pass Argon2i since it runs twice as fast and consumes the same memory. In practice, this may not always be the case. To compute each new label in iXOR we need to load 20 new $1KB$ blocks from memory and XOR them before applying the hash function. For Argon2ib we only need to load 1 new $1KB$ block from memory before applying the hash function. If memory bandwidth is not a bottleneck then 3-pass iXOR may be preferable to 6-pass Argon2i. Otherwise, 6-pass Argon2i may actually run faster than 3-pass iXOR.

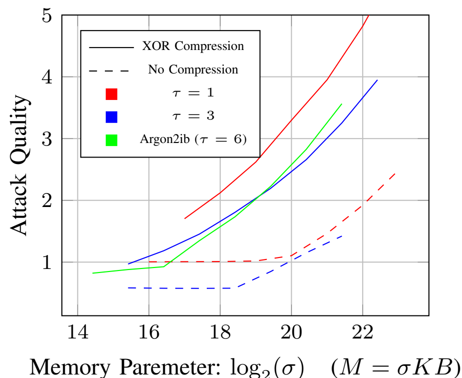

Figure 7: $iXOR$ Attack Quality (indeg = 21)

19. The newest version of the Balloon Hashing algorithm does apply Merkle-Damgard, but because the newest Balloon Hashing algorithm has indeg = 3 slowdown is less of an issue.

20. It is possible that we won't have all of these parent labels available when the balloon phase finishes (i.e., because some of v's parents will be pebbled for the first time in the next light phase). However, this is not a problem as we can simply store the XOR of all known parent labels and XOR this block with the remaining parent label(s) as they become available in during the light phase.