Bandwidth Hard Functions for ASIC Resistance

Ling Ren(✉) and Srinivas Devadas

Massachusetts Institute of Technology, Cambridge, MA, USA
{renling,devadas}@mit.edu

Abstract. Cryptographic hash functions have wide applications including password hashing, pricing functions for spam and denial-of-service countermeasures and proof of work in cryptocurrencies. Recent progress on ASIC (Application Specific Integrated Circuit) hash engines raise concerns about the security of the above applications. This leads to a growing interest in ASIC resistant hash function and ASIC resistant proof of work schemes, i.e., those that do not give ASICs a huge advantage. The standard approach towards ASIC resistance today is through memory hard functions or memory hard proof of work schemes. However, we observe that the memory hardness approach is an incomplete solution. It only attempts to provide resistance to an ASIC's area advantage but overlooks the more important energy advantage. In this paper, we propose the notion of bandwidth hard functions to reduce an ASIC's energy advantage. CPUs cannot compete with ASICs for energy efficiency in computation, but we can rely on memory accesses to reduce an ASIC's energy advantage because energy costs of memory accesses are comparable for ASICs and CPUs. We propose a model for hardware energy cost that has sound foundations in practice. We then analyze the bandwidth hardness property of ASIC resistant candidates. We find scrypt, Catena-BRG and Balloon are bandwidth hard with suitable parameters. Lastly, we observe that a capacity hard function is not necessarily bandwidth hard, with a stacked double butterfly graph being a counterexample.

1 Introduction

Cryptographic hash functions have a wide range of applications in both theory and practice. Two of the major applications are password protection and more recently proof of work. It is well known that service providers should store hashes of user passwords. This way, when a password hash database is breached, an adversary still has to invert the hash function to obtain user passwords. Proof of work, popularized by its usage in the Bitcoin cryptocurrency for reaching consensus [43], has earlier been used as "pricing functions" to defend against email spam and denial-of-service attacks [19,30].

In the last few years, driven by the immense economic incentives in the Bitcoin mining industry, there has been amazing progress in the development of ASIC (Application Specific Integrated Circuit) hash units. These ASIC hash engines are specifically optimized for computing SHA-256 hashes and offer incredible speed and energy efficiency that CPUs cannot hope to match. A state-of-the-art ASIC Bitcoin miner [1] computes 13 trillion hashes at about 0.1 nJ energy cost per hash. This is roughly 200,000× faster and 40,000× more energy efficient than a state-of-the-art multi-core CPU. These ASIC hash engines call the security of password hashing and pricing functions into question. For ASIC-equipped adversaries, brute-forcing a password database seems quite feasible, and pricing functions are nowhere near deterrent if they are to stay manageable for honest CPU users. ASIC mining also raises some concerns about the decentralization promise of Bitcoin as mining power concentrates to ASIC-equipped miners.

As a result, there is an increasing interest in ASIC resistant hash functions and ASIC resistant proof of work schemes, i.e., those that do not give ASICs a huge advantage. For example, in the recent Password Hashing Competition [34], the winner Argon2 [21] and three of the four "special recognitions" — Catena [35], Lyra2 [9] and yescrypt [47] — claimed ASIC resistance. More studies on ASIC resistant hash function and proof of work include [10–15,18,23,25,39,46,49,53].

The two fundamental advantages of ASICs over CPUs (or general purpose GPUs) are their smaller area and better energy efficiency when speed is normalized. The speed advantage can be considered as a derived effect of the area and energy advantage (cf. Sect. 3.1). A chip's area is approximately proportional to its manufacturing cost. From an economic perspective, this means when we normalize speed, an adversary purchasing ASICs can lower its initial investment (capital cost) due to area savings and its recurring electricity cost due to energy savings, compared to a CPU user. To achieve ASIC resistance is essentially to reduce ASICs' area and energy efficiency advantages.

Most prior works on ASIC resistance have thus far followed the *memory hard function* approach, first proposed by Percival [46]. This approach tries to find functions that require a lot of memory *capacity* to evaluate. To better distinguish from other notions later in the paper, we henceforth refer to memory hard functions as *capacity hard functions*. For a traditional hash function, an ASIC has a big area advantage because one hash unit occupies much smaller chip area than a whole CPU. The reasoning behind a capacity hard function is to reduce an ASIC's area advantage by forcing it to spend significant area on memory. Historically, the capacity hardness approach only attempts to resist the area advantage. We quote from Percival's paper [46]:

> A natural way to reduce the advantage provided by an attacker's ability to construct highly parallel circuits is to increase the size of the key derivation circuit — if a circuit is twice as large, only half as many copies can be placed on a given area of silicon ...

Very recently, some works [10,11,14] analyze capacity hard functions from an energy angle (though they try to show negative results). However, an energy model based on memory capacity cannot be justified from a hardware perspective. We defer a more detailed discussion to Sects. 6.2 and 6.3.

It should now be clear that the capacity hardness approach does not provide a full solution to ASIC resistance since it only attempts to address the area aspect, but not the energy aspect of ASIC advantage. Fundamentally, the relative importance of these two aspects depends on many economic factors, which is out of the scope of this paper. But it may be argued that the energy aspect is more important than the area aspect in many scenarios. Area advantage, representing lower capital cost, is a one-time gain, while energy advantage, representing lower electricity consumption, keeps accumulating with time. The goal of this paper is to fill in the most important but long-overlooked energy aspect of ASIC resistance.

1.1 Bandwidth Hard Functions

We hope to find a function f that ensures the energy cost to evaluate f on an ASIC cannot be much smaller than on a CPU. We cannot change the fact that ASICs have much superior energy efficiency for computation compared to CPUs. Luckily, to our rescue, off-chip memory accesses incur comparable energy costs on ASICs and CPUs, and there are reasons to believe that it will remain this way in the foreseeable future (cf. Sect. 6.1). Therefore, we would like an ASIC resistant function f to be *bandwidth hard*, i.e., it requires a lot of off-chip memory accesses to evaluate f. Informally, if off-chip memory accesses account for a significant portion of the total energy cost to evaluate f, it provides an opportunity to bring the energy cost on ASICs and CPUs onto a more equal ground.

A capacity hard function is not necessarily bandwidth hard. Intuitively, an exception arises when a capacity hard function has good locality in its memory access pattern. In this case, an ASIC adversary can use some on-chip cache to "filter out" many off-chip memory accesses. This makes computation the energy bottleneck again and gives ASICs a big advantage in energy efficiency. A capacity hard function based on a stacked double butterfly graph is one such example (Sect. 5.4).

On the positive side, most capacity hard functions are bandwidth hard. Scrypt has a data-dependent and (pseudo-)random memory access pattern. A recent work shows that scrypt is also capacity hard even under amortization and parallelism [13]. Adapting results from the above work, we prove scrypt is also bandwidth hard in Sect. 5.1 with some simplifying assumptions. Thus, scrypt offers nearly optimal ASIC resistance from both the energy aspect and the area aspect. But scrypt still has a few drawbacks. First, scrypt is bandwidth hard only when its memory footprint (i.e., capacity requirement) is much larger than the adversary's cache size. In practice, we often see protocol designers adopt scrypt with too small a memory footprint (to be less demanding for honest users) [5], which completely undermines its ASIC resistance guarantee [6]. Second, in password hashing, a data-dependent memory access pattern is considered to be less secure for fear of side channel attacks [25]. Thus, it is interesting to also look for *data-independent* bandwidth hard functions, especially those that achieve bandwidth hardness with a smaller memory footprint.

To study data-independent bandwidth hard functions, we adopt the graph labeling framework in the random oracle model, which is usually modeled by the pebble game abstraction. The most common and simple pebble game is the black pebble game, which is often used to study space and time complexity. To model the cache/memory architecture that an adversary may use, we adopt the red-blue pebble game [33,50]. In a red-blue game, there are two types of pebbles. A red (hot) pebble models data in cache and a blue (cold) pebble models data in memory. Data in memory must be brought into the cache before being computed on. Accordingly, a blue pebble must be "turned into" a red pebble (i.e., brought into the cache) before being used by a computation unit. We incorporate an energy cost model into red-blue pebble games, and then proceed to analyze data-independent bandwidth hard function candidates. We show that Catena-BRG [35] and Balloon [25] are bandwidth hard in the pebbling model. But we could not show a reduction from graph labeling with random oracles to red-blue pebble games. Thus, all results on data-independent graphs are only in the pebbling mode. A reduction from labeling to pebbling remains interesting future work.

Our idea of using memory accesses resembles, and is indeed inspired by, a line of work called memory bound functions [8,29,31]. Memory bound functions predate capacity hard functions, but unfortunately have been largely overlooked by recent work on ASIC resistance. We mention one key difference between our work and memory bound functions here and defer a more detailed comparison in Sect. 2. Memory bound functions assume computation is free for an adversary and thus aim to put strict lower bounds on the number of memory accesses. We, on the other hand, assume computation is cheap but not free for an adversary (which we justify in Sect. 4.1). As a result, we just need to guarantee that an adversary who attempts to save memory accesses has to compensate with so much computation that it ends up increasing its energy consumption. This relaxation of "bandwidth hardness" leads to much more efficient and practical solutions than existing memory bound functions [8,29,31]. To this end, the term "bandwidth hard" and "memory hard" may be a little misleading as they do not imply strict lower bounds on bandwidth and capacity. Memory (capacity) hardness as defined by Percival [46] refers to a lower bound on the space-time product ST, while bandwidth hardness in this paper refers to a lower bound on an ASICs' energy consumption under our model.

1.2 Our Contributions

We observe that energy efficiency, as the most important aspect of ASIC resistance, has thus far not received much attention. To this end, we propose using bandwidth hard functions to reduce the energy advantage of ASICs. We propose a simple energy model and incorporate it into red-blue pebble games. We note that ASIC resistance is a complex real-world notion that involves low-level hardware engineering. Therefore, in this paper we go over the reasoning and basic concepts of ASIC resistance from a hardware perspective and introduce a model based on hardware architecture and energy cost in practice.

Based on the model, we study the limit of ASIC energy resistance. Roughly speaking, an ASIC adversary can always achieve an energy advantage that equals the ratio between a CPU's energy cost per random oracle evaluation and an ASIC's energy cost per memory access. We observe that if we use a hash function (e.g., SHA-256) as the random oracle, which is popular among ASIC resistant proposals, it is impossible to reduce an ASIC's energy advantage below $100\times$ in today's hardware landscape. Fortunately, we may be able to improve the situation utilizing CPUs' AES-NI instruction extensions.

We then turn our attention to analyzing the bandwidth hardness properties of ASIC resistant candidate constructions. We prove in the pebbling model that scrypt [46], Catena-BRG [35] and Balloon [25] enjoy tight bandwidth hardness under suitable parameters. Lastly, we point out that a capacity hard function is not necessarily bandwidth hard, using a stacked double butterfly graph as a counterexample.

2 Related Work

Memory (capacity) hard functions. Memory (capacity) hard functions are currently the standard approach towards ASIC resistance. The notion was first proposed by Percival [46] along with the scrypt construction. There has been significant follow-up that propose constructions with stronger notions of capacity hardness [9,15,21,25,35,39,49]. As we have noted, capacity hardness only addresses the area aspect of ASIC resistance. It is important to consider the energy aspect for a complete solution to ASIC resistance.

Memory (capacity) hard proof of work. Memory (capacity) hard proofs of work [18,23,49,53] are proof of work schemes that require a prover to have a lot of memory capacity, but at the same time allow a verifier to check the prover's work with a small amount of space and time. The motivation is also ASIC resistance, and similarly, it overlooks the energy aspect of ASIC resistance.

Memory bound functions. The notion of memory bound functions was first proposed by Abadi et al. [8] and later formalized and improved by Dwork et al. [29,31]. A memory bound function requires a lot of memory accesses to evaluate. Those works do not relate to an energy argument, but rather use speed and hence memory latency as the metrics. As we discuss in Sect. 3.1, using speed as the only metric makes it hard to interpret the security guarantee in a normalized sense. Another major difference is that memory bound functions assume computation is completely free and aim for strict lower bounds on bandwidth (the number of memory accesses), while we assume computation is cheap but not free. To achieve its more ambitious goal, memory bound function constructions involve traversing random paths in a big table of true random numbers. This results in a two undesirable properties. First, the constructions are inherently data-dependent, which raises some concerns for memory access pattern leakage in password hashing. Second, the big random table needs to be transferred over the network between a prover (who computes the function) and a verifier (who checks the prover's computation). A follow-up work [31] allows the big table to be filled by pebbling a rather complex graph (implicitly moving to our model where computation is cheap but not free), but still relies on the random walk in the table to enforce memory accesses. Our paper essentially achieves the same goal just through pebbling and from simpler graphs, thus eliminating the random walk and achieving better efficiency and data-independence.

Parallel attacks. An impressive recent line of work has produced many interesting results regarding capacity hardness in the presence of parallel attacks. These works show that a parallel architecture can reduce the area-time product for any data independent capacity hard function [10,12,15]. The practical implications of these attacks are less clear and we defer a discussion to Sect. 6.3. We would also like to clarify a direct contradiction between some parallel attacks' claims [10,11,14] and our results. We prove that Catena-BRG [35] and Balloon [25] enjoy great energy advantage resistance while those works conclude the exact opposite. The contradiction is due to their energy model that we consider flawed, which we discuss in Sect. 6.3.

Graph pebbling. Graph pebbling is a powerful tool in computer science, dating back at least to 1970s in studying Turing machines [27,36] and register allocation [51]. More recently, graph pebbling has found applications in various areas of cryptography [18,25,31–33,35,40,49,52]. Some of our proof techniques are inspired by seminal works in pebbling lower bounds and trade-offs by Paul and Tarjan [44] and Lengauer and Tarjan [38].

3 Preliminaries

3.1 A Hardware Perspective on ASIC Resistance

The first and foremost question we would like to answer is: what advantages of ASICs are we trying to resist? The most strongly perceived advantage of ASIC miners may be their incredible speed, which can be a million times faster than CPUs [1]. But if speed were the sole metric, we could just deploy a million CPUs in parallel to catch up on speed. Obviously, using a million CPUs would be at a huge disadvantage in two aspects: capital cost (or manufacturing cost) and power consumption. The manufacturing cost of a chip is often approximated by its area in theory [41]. Therefore, the metrics to compare hardware systems should be:

1. the area-speed ratio, or equivalently the area-time product, commonly referred to as AT in the literature [10,11,22,41], and
2. the power-speed ratio, which is equivalent to energy cost per function evaluation.

Area and energy efficiency are the two major advantages of ASICs. The speed advantage can be considered as a derived effect from them. Because an ASIC hash unit is small and energy efficient, ASIC designers can pack thousands of them in a single chip and still have reasonable manufacturing cost and manageable power consumption and heat dissipation.

Bandwidth hard functions to address both aspects. Percival proposes using capacity hard functions to reduce ASIC area advantage [46]. With bandwidth hard functions, we hope to additionally reduce ASIC's energy advantage. We note that a bandwidth hard function also needs to be capacity hard. Thus, a hardware system evaluating it, be it a CPU or an ASIC, needs off-chip external memory. This has two important implications. First, a bandwidth hard function inherits the area advantage resistance from capacity hardness (though somewhat weakened by parallel attacks). Second, a bandwidth hard function forces an ASIC into making a lot of off-chip memory accesses, which limits the ASIC's energy advantage. To study hardware energy cost more formally, we need to introduce a hardware architecture model and an energy cost model.

Hardware architecture model. The adversary is allowed to have any cache policy on its ASIC chip, e.g., full associativity and optimal replacement [20]. Our proofs do not directly analyze cache hit rate, but the results imply that a bandwidth hard function ensures a low hit rate even for an optimal cache. We assume a one-level cache hierarchy for convenience. This does not affect the accuracy of the model. We do not charge the adversary for accessing data from the cache, so only the total cache size matters. Meanwhile, although modern CPUs are equipped with large caches, honest users cannot utilize it since a bandwidth hard function has very low cache hit rate. We simply assume a 0% cache hit rate for honest users.

Energy cost model. We assume it costs c_b energy to transfer one bit of data between memory and cache, and c_r energy to evaluate the random oracle on one bit of data in cache. If an algorithm transfers B bits of data and queries the random oracle on R bits of data in total, its energy cost is $\mathsf{ec} = c_b B + c_r R$. A compute unit and memory interface may operate at a much larger word granularity, but we define c_b and c_r to be amortized per bit for convenience. The two coefficients are obviously hardware dependent. We write $c_{b,\mathsf{cpu}}, c_{r,\mathsf{cpu}}$ and $c_{b,\mathsf{asic}}, c_{r,\mathsf{asic}}$ when we need to distinguish them. The values of these coefficients are determined experimentally or extracted from related studies or sources in Sect. 4.1. Additional discussions and justifications of the models are presented in Sect. 6.

Energy fairness. Our ultimate goal is to achieve *energy fairness* between CPUs and ASICs. For a function f, suppose honest CPU users adopt an algorithm with an energy cost $\mathsf{ec}_0 = c_{b,\mathsf{cpu}} B_0 + c_{r,\mathsf{cpu}} R_0$. Let $\overline{\mathsf{ec}} = \overline{\mathsf{ec}}(f, M, c_{b,\mathsf{asic}}, c_{r,\mathsf{asic}})$ be the minimum energy cost for an adversary to evaluate f with cache size M and ASIC energy parameters $c_{b,\mathsf{asic}}$ and $c_{r,\mathsf{asic}}$. Energy fairness is then measured by the energy advantage of an ASIC adversary over honest CPU users (under those parameters): $A_{\mathsf{ec}} = \mathsf{ec}_0/\overline{\mathsf{ec}} > 1$. A smaller A_{ec} indicates a smaller energy advantage of ASICs, and thus better energy fairness between CPUs and ASICs.

We remark that while an ASIC's energy cost for computation $c_{r,\mathsf{asic}}$ is small, we assume it is not strictly 0. It is assumed in some prior works that computation is completely free for the adversary [31,33]. In that model, we must find a function that simultaneously satisfies the following two conditions: (1) it has a trade-off-proof space lower bound (even an exponential computational penalty for space reduction is insufficient), and (2) it requires a comparable amount of computation and memory accesses. We do not know of any candidate data-independent construction that satisfies both conditions. We believe our assumption of a non-zero $c_{r,\mathsf{asic}}$ is realistic, and we justify it in Sect. 4.1 with experimental values. In fact, our model may still be overly favorable to the adversary. It has been noted that the energy cost to access data from an on-chip a cache is roughly proportional to the square root of the cache size [16]. Thus, if an adversary employs a large on-chip cache, the energy cost of fetching data from this cache needs to be included in $c_{r,\mathsf{asic}}$.

3.2 The Graph Labeling and Pebbling Framework

We adopt the graph labeling and pebbling framework that is common in the study of ASIC resistant [15,18,25,31,35].

Graph labeling. Graph labeling is a computational problem that evaluates a random oracle $\mathcal{H}$ in a directed acyclic graph (DAG) G. A vertex with no incoming edges is called a *source* and a vertex with no outgoing edges is called a *sink*. Vertices in G are numbered, and each vertex v_i is associated with a label $l(v_i)$, computed as:

$$l(v_i) = \begin{cases} \mathcal{H}(i, x) & \text{if } v_i \text{ is a source} \\ \mathcal{H}(i, l(u_1), \cdots, l(u_d)) & \text{otherwise, } u_1 \text{ to } u_d \text{ are } v_i\text{'s predecessors} \end{cases}$$

The output of the graph labeling problem are the labels of the sinks. It is common to hash the labels of all sinks into a final output (of the ASIC resistant function) to keep it short.

Adversary. We consider a deterministic adversary that has access to $\mathcal{H}$ that runs in rounds, starting from round 1. In round i, the adversary receives an input state σ_i and produces an output state $\bar{\sigma}_i$. Each input state $\sigma_i = (\tau_i, \eta_i, h_i)$ consists of

- τ_i, M bits of data in cache,
- η_i, an arbitrary amount of data in memory, and
- a w-bit random oracle response h_i if a random oracle query was issued in the previous round.

Each output state $\bar{\sigma}_i = (\bar{\tau}_i, \eta_i, q_i)$ consists of

- $\bar{\tau}_i$, M bits of data in cache, which can be any deterministic function of σ_i and h_i.
- η_i, data in memory, which is unchanged from the input state, and
- an operation q_i that can either be a random oracle query or a data transfer operation between memory and cache.

If the operation q_i is a random oracle query, then in the input state of the next round, the random oracle response is $h_{i+1} = \mathcal{H}(q_i)$ and the contents of the cache/memory is unchanged $(\sigma_{i+1}, \eta_{i+1}) = (\bar{\sigma}_i, \eta_i)$. If the operation q_i is a data transfer operation, then it has the form (x_i, y_i, z_i, b_i) in which x_i is an offset in the cache, y_i is an offset in memory, z_i specifies whether the direction of the transfer is from cache to memory ($z_i = 0$) or from memory to cache ($z_i = 1$), and b_i is the number of bits to be transfered. In the input state of the next round, the contents of the cache/memory $(\sigma_{i+1}, \eta_{i+1})$ are obtained by applying the data transfer operation on $(\bar{\sigma}_i, \eta_i)$, and $h_{i+1} = \perp$. The energy cost of the adversary is defined as follows. A random oracle call on $r_i = |q_i|$ bits if input costs $c_r r_i$ units of energy. A data transfer of b_i bits in either direction costs $c_b b_i$ units of energy. Any other computation that happens during a round is free for the adversary. The total energy cost of the adversary is the sum of cost in all rounds.

Pebble games. Graph labeling is often abstracted as a pebble game. Computing $l(v)$ is modeled as placing a pebble on vertex v. The goal of the pebble game in our setting is to place pebbles on the sinks. There exist several variants of pebble games. The simplest one is the *black* pebble game where there is only one type of pebbles. In each move, a pebble can be placed on vertex v if v is a source or if all predecessors of v have pebbles on them. Pebbles can be removed from any vertices at any time.

Red-blue pebble games. To model a cache/memory hierarchy, *red-blue* pebble games have been proposed [33,50]. In this game, there are two types of pebbles. A red (hot) pebble models data in cache, which can be computed upon immediately. A blue (cold) pebble models data in memory, which must first be brought into cache to be computed upon. The rule of a red-blue pebble game is naturally extended as follows:

1. A *red* pebble can be placed on vertex v if v is a source or if all predecessors of v have *red* pebbles on them.
2. A red pebble can be placed on vertex v if there is a blue pebble on v. A blue pebble can be placed on vertex v if there is a red pebble on v.

We refer to the first type of moves as *red moves* and the second type as *blue moves*. Pebbles (red or blue) can be removed from any vertices at any time. A pebbling strategy can be represented as a sequence of transitions between pebble placement configurations on the graph, $\mathbf{P} = (P_0, P_1, P_2 \cdots, P_T)$. Each configuration consists of two vectors of size $|V|$, specifying for each vertex if a red pebble exists and if a blue pebble exists. The starting configuration P_0 does not have to be empty; pebbles may exist on some vertices in P_0. Each transition makes *either* a red move *or* a blue move, and then removes any number of pebbles for free.

We introduce some extra notations. If a pebble (red or blue) exists on a vertex v in a configuration P_i, we say v is *pebbled* in P_i. We say a sequence $\mathbf{P}$ pebbles a vertex v if there exists $P_i \in \mathbf{P}$ such that v is pebbled in P_i. We say a sequence $\mathbf{P}$ pebbles a set of vertices if $\mathbf{P}$ pebbles every vertex in the set. Note that blue pebbles cannot be directly created on unpebbled vertices. If a vertex v is not initially pebbled in P_0, then the first pebble that gets placed on v in $\mathbf{P}$ must be a red pebble, and it must result from a red move.

Energy cost of red-blue pebbling. In red-blue pebbling, red moves model computation on data in cache and blue moves model data transfer between cache and memory. It is straightforward to adopt the energy cost model in Sect. 3.1 to a red-blue pebbling sequence $\mathbf{P}$. We charge c_b cost for each blue move. For each red move (i.e., random oracle call), we charge a cost proportional to the number of input vertices. Namely, if a vertex v has d predecessors, a red move on v costs $c_r d$ units of cost. Similarly, we write $c_{b,\mathsf{cpu}}, c_{r,\mathsf{cpu}}$ and $c_{b,\mathsf{asic}}, c_{r,\mathsf{asic}}$ when we need to distinguish them. The energy coefficients in Sect. 3.1 are defined per bit and here they are per label. This is not a problem because only the ratio between these coefficients matter. As before, removing pebbles (red or blue) is free. If $\mathbf{P}$ uses B blue moves and R red moves each with d predecessors, it incurs a total energy cost $\mathsf{ec}(\mathbf{P}) = c_b B + c_r d R$.

The adversary's cache size M translates to a bounded number of red pebbles at any given time, which we denote as m. For a graph G, given parameters m, c_b and c_r, let $\overline{\mathsf{ec}} = \overline{\mathsf{ec}}(G, m, c_b, c_r)$ be the minimum cost to pebble G in a red-blue pebble game starting with an empty initial configuration under those parameters. Let ec_0 be the cost of an honest CPU user. The energy advantage of an ASIC is $A_{\mathsf{ec}} = \mathsf{ec}_0/\overline{\mathsf{ec}}$.

Definition of bandwidth hardness. The ultimate goal of a bandwidth hard function is to achieve fairness between CPUs and ASICs in terms of energy cost. In the next section, we will establish $\overline{A_{\mathsf{ec}}} = \frac{c_{b,\mathsf{cpu}} + c_{r,\mathsf{cpu}}}{c_{b,\mathsf{asic}} + c_{r,\mathsf{asic}}}$ as a lower bound on the adversary's energy advantage A_{ec} for any function. We say a function *under a particular parameter setting* is bandwidth hard if it ensures $A_{\mathsf{ec}} = \Theta(\overline{A_{\mathsf{ec}}})$, i.e., if it upper bounds an adversary's energy advantage to a constant within the best we can hope for.

In the above definition, we emphasize "under a particular parameter setting" because we will frequently see that a function's bandwidth hardness kicks in only when its memory capacity requirement n is sufficiently large compared to the adversary's cache size m. This should be as expected: if the entire memory footprint fits in cache, then a function must be computation bound rather than bandwidth bound. As an example, we will later show that scrypt is bandwidth hard when it requires sufficiently large memory capacity. But when scrypt is adopted in many practical systems (e.g., Litecoin), it is often configured to use much smaller memory, thus losing its bandwidth hardness and ASIC resistance.

Connection between labeling and pebbling. The labeling-to-pebbling reduction has been established for data-independent graphs [31,33] and for scrypt [13] when the metric is space complexity or cumulative complexity. Unfortunately, for bandwidth hard functions and energy cost, we do yet know how to reduce the graph labeling problem with a cache to the red-blue pebbling game without making additional assumptions. The difficulty lies in how to transform the adversary's data transfer between a memory and a cache into blue moves. Thus, all results for data-independent graphs in this paper will be in the red-blue pebbling model. This is equivalent to placing a restriction on the adversary that it can only transfer whole labels between cache and memory. Showing a reduction for data-independent graphs without the above restriction is an interesting open problem. We mention that for general data-dependent graphs and proofs of space [32], a reduction from labeling to black pebbling also remains open.

4 The Limit of Energy Fairness

While our goal is to upper bound the energy advantage A_E, it is helpful to first look at a lower bound to know how good a resistance we can hope for. Suppose honest users adopt an algorithm that transfers B_0 bits and queries $\mathcal{H}$ on R_0 bits in total. Even if an adversary does not have a better algorithm, it can simply adopt the honest algorithm but implements it on an ASIC. In this case, the adversary's energy advantage is

$$A_{\mathsf{ec}} = \frac{c_{b,\mathsf{cpu}} B_0 + c_{r,\mathsf{cpu}} R_0}{c_{b,\mathsf{asic}} B_0 + c_{r,\mathsf{asic}} R_0} = \frac{c_{b,\mathsf{cpu}} + c_{r,\mathsf{cpu}} R_0/B_0}{c_{b,\mathsf{asic}} + c_{r,\mathsf{asic}} R_0/B_0}.$$

Since we expect $c_{r,\mathsf{cpu}} \gg c_{r,\mathsf{asic}}$ and $c_{b,\mathsf{cpu}} \approx c_{b,\mathsf{asic}}$, the above value is smaller when R_0/B_0 is smaller (more memory accesses and less computation). Any data brought into the cache must be consumed by the compute unit (random oracle) — otherwise, the data transfer is useless and should not have happened. Given that $B_0 \leq R_0$, the adversary can at least have an energy consumption advantage of:

$$A_{\mathsf{ec}} \geq \frac{c_{b,\mathsf{cpu}} + c_{r,\mathsf{cpu}}}{c_{b,\mathsf{asic}} + c_{r,\mathsf{asic}}} = \overline{A_{\mathsf{ec}}}.$$

In Sects. 5.2 and 5.3, we prove that bit reversal graphs and stacked expanders essentially reduce A_{ec} very close to the lower bound $\overline{A_{\mathsf{ec}}}$. So $\overline{A_{\mathsf{ec}}}$ is quite tight and represents both the lower and upper limit of the energy advantage resistance we can achieve.

Since we expect $c_{r,\mathsf{asic}}$ to be small, and $c_{b,\mathsf{cpu}} \approx c_{b,\mathsf{asic}}$, the above lower bound is approximately $1 + c_{r,\mathsf{cpu}}/c_{b,\mathsf{cpu}}$. So we hope $c_{r,\mathsf{cpu}}$ to be small and $c_{b,\mathsf{cpu}}$ to be large, in which case memory accesses account for a significant portion of the total energy cost on CPUs. It is often mentioned that computation is cheap compared to memory accesses even for CPUs, which seems to be in our favor. However, the situation is much less favorable for our scenario because a cryptographic hash is a complex function that involves thousands of operations. It would be unrealistic for us to assume $c_{r,\mathsf{cpu}} \ll c_{b,\mathsf{cpu}}$. To estimate the concrete value of $\overline{A_{\mathsf{ec}}}$, in Sect. 4.1 we conduct experiments to measure $c_{r,\mathsf{cpu}}$ and $c_{b,\mathsf{cpu}}$ and cite estimates of $c_{r,\mathsf{asic}}$ and $c_{b,\mathsf{asic}}$ from reliable sources.

4.1 Experiments to Estimate Energy Cost Coefficients

All values we report here are approximates as their exact values depend on many low level factors (technology process, frequency, voltage, etc.). Nevertheless, they should allow us to estimate A_{ec} to the correct order of magnitude.

We keep a CPU fully busy with the task under test, i.e., compute hashes and making memory accesses. We use Intel Power Gadget [4] to measure the CPU package energy consumption in a period of time, and then divide by the number of Bytes processed (hashed or transferred). We run tests on an Intel Core I7-4600U CPU in 22 nm technology clocked at 1.4 GHz. The operating system is Ubuntu 14.04 and we use Crypto++ Library 5.6.3 compiled with GCC 4.6.4.

Table 1. Measured energy cost (in nJ) per Byte for memory accesses and cryptographic operations on CPUs.

Operation	Memory access	SHA-256	AES-NI
Energy, CPU	0.5	30	1.5
Energy, ASIC	0.3	0.0012	/

Table 1 reports the measured CPU energy cost per Bytes. For comparison, we take the memory access energy estimates for ASICs from two papers [37, 45], which have very close estimations. We take the SHA-256 energy cost for ASIC from the state-of-the-art Antminer S9 specification [1]. Antminer S9 spends 0.098 nJ to hash 80 Bytes, which normalizes to 0.0012 nJ/Byte.

4.2 Better Energy Fairness with AES-NI

From the above results, we have $c_{b,\mathsf{cpu}} \approx 0.5$, $c_{b,\mathsf{asic}} \approx 0.3$, and if we use SHA-256 to implement the random oracle $\mathcal{H}$, then $c_{r,\mathsf{cpu}} \approx 30$ and $c_{r,\mathsf{asic}} \approx 0.1$. With these parameters, any function in the graph labeling framework can at most reduce an ASIC's energy advantage to $\overline{A_{\mathsf{ec}}} \approx (0.5 + 30)/(0.3 + 0.0012) \approx 100\times$. While this represents an improvement over plain SHA-256 hashing (which suffers from an energy advantage of roughly $30/0.0012 = 25,000\times$), $100\times$ is still a quite substantial advantage.

Is $100\times$ the limit of energy fairness or can we do better? To push $\overline{A_{\mathsf{ec}}}$ lower, we need a smaller $c_{r,\mathsf{cpu}}$. The AES-NI extension gives exactly what we need. AES-NI

(AES New Instructions) [3] is a set of new CPU instructions specifically designed to improve the speed and energy efficiency of AES operations on CPUs. Today AES-NI is available in all mainstream Intel processors. In fact, AES-NI is an ASIC-style AES circuit that Intel builds into its CPUs, which is why it reduces ASIC advantage. But also we cannot expect AES-NI to completely match standalone AES ASICs because it is subject to many design constrains imposed by Intel CPUs.

We repeat our previous experiments to measure the energy efficiency of AES operations on CPUs. As expected, AES-NI delivers much better energy efficiency, 1.5 nJ per Byte. We do not know for sure what $c_{r,\mathsf{asic}}$ would be for AES, but expect it to be no better than SHA-256 (and the bounds are insensitive to $c_{r,\mathsf{asic}}$ since $c_{b,\mathsf{asic}}$ dominates in the denominator). Therefore, if we use AES for pebbling, the lower bound drops to $\overline{A_{\mathsf{ec}}} \approx (0.5 + 1.5)/0.3 \approx 6.7\times$. It is worth noting that using AES for pebbling also reduces an ASIC's AT advantage as it makes CPUs run faster (smaller T).

Great care needs to be taken when instantiating the random oracle with a concrete function. Boneh et al. [25] point out that the pebbling analogy breaks down if the random oracle $\mathcal{H}$ is instantiated with a cryptographic hash function based on the Merkle-Damgård construction [28,42]. The problem is that a Merkle-Damgård construction does not require its entire input to be present at the same time, but instead absorbs the input chunk by chunk. The same caveat exists when we use AES for pebbling. We leave a thorough study on pebbling with AES to future work. If we want even smaller $c_{r,\mathsf{cpu}}$ and $\overline{A_{\mathsf{ec}}}$ or to avoid the complication of using AES, we may have to count on Intel's SHA instruction extensions. Intel announced plans to add SHA extensions a few years ago [7], but no product has incorporated them so far.

5 Bandwidth Hardness of Candidate Constructions

Some candidate constructions we analyze in this section are based on a class of graphs called "sandwich graphs" [15,25]. A sandwich graph is a directed acyclic graph $G = (V \cup U, E)$ that has $2n$ vertices $V \cup U = (v_0, v_1, \cdots v_{n-1}) \cup (u_0, u_1, \cdots u_{n-1})$, and two types of edges:

- chain edges, i.e., (v_i, v_{i+1}) and (u_i, u_{i+1}) $\forall i \in [0..n-2]$, and
- cross edges from V to U.

Figure 1 is a random sandwich graph with $n = 8$. In other words, a sandwich graph is a bipartite graph with the addition of chain edges. We call the path consisting of $(v_0, v_1), (v_1, v_2), \cdots (v_{n-2}, v_{n-1})$ the *input path*, and the path consisting of $(u_0, u_1), (u_1, u_2), \cdots (u_{n-2}, u_{n-1})$ the *output path*.

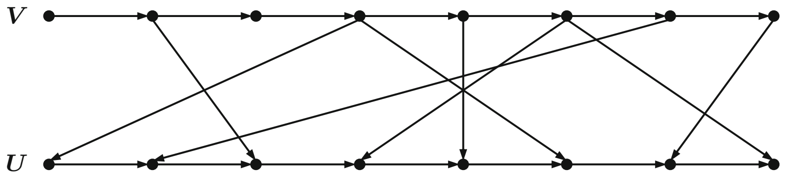

Fig. 1. A random sandwich graph with $n = 8$.

5.1 Scrypt

Scrypt [46] can be thought of as a sandwich graph where the cross edges are dynamically generated at random in a data-dependent fashion. Each vertex u_i on the output path has one incoming cross edge from a vertex v_j that is chosen uniformly random from the input path based on the previous label $l(u_{i-1})$ (or $l(v_{n-1})$ for u_0), and thus cannot be predicted beforehand.

The default strategy to compute scrypt is to first compute each $l(v_i)$ on the input path sequentially and store each one in memory, and then compute each $l(u_i)$ on the output path, fetching each $l(v_j)$ from memory as needed. The total cost of this strategy is $(c_r + c_b)n + (c_b + 2c_r)n = (2c_b + 3c_r)n$ (every node on the output path has in-degree 2).

To lower bound the energy cost, we make a simplifying assumption that if the adversary transfers data from memory to cache at all, it transfers at least w bits where $w = |l(\cdot)|$ is the label size. We also invoke the "single-challenge time lower bound" theorem on scrypt [13], which we paraphrase below. The adversary can fill a cache of M bits after arbitrary computation and preprocessing on the input path. The adversary then receives a challenge j chosen uniformly at random from 0 to $n - 1$ and tries to find $l(v_j)$ using only the data in the cache. Let t be the random variable that represents the number of *sequential* random oracle calls to $\mathcal{H}$ made by the adversary till it queries $\mathcal{H}$ with $l(v_j)$ for the first time.

Theorem 1 (Alwen et al. [13]). *For all but a negligible fraction of random oracles, the following holds: given a cache of M bits, $\Pr[t > \frac{n}{2p}] > \frac{1}{2}$ where $p = (M + 1)/(w - 3\log n + 1)$ and $w = |l(\cdot)|$ is the label size.*

The above theorem states that in the parallel random oracle model, with at least 1/2 probability, an adversary needs $n/2p$ sequential random oracles to answer the random challenge. (Note that the above theorem does not directly apply to scrypt, since challenges in scrypt come from the random oracle rather than from an independent external source. This issue can be handled similarly as in [13].) A lower bound on the number of sequential random oracle calls in the parallel model is also a lower bound on the number of total random oracle calls in our sequential model. Theorem 1 states that if the adversary wishes to compute a label on the output path only using the M bits in cache without fetching from memory, there is a 1/2 chance that doing so requires $n/2p$ random oracle calls. If we choose a sufficiently large n such that $c_r n/2p > c_b$, then making $n/2p$

random oracle calls is more expensive than simply fetching the challenged input label from memory. Since we assume the adversary fetches w bits at a time, so if it fetches from memory at all, it rather fetches the challenged input label. Then, for any adversary, the expected cost to compute a label on the output path is at least $c_b/2$ and the energy advantage is at most $A_{\mathsf{ec}} < \frac{2c_{b,\mathsf{cpu}}+3c_{r,\mathsf{cpu}}}{0.5c_{b,\mathsf{asic}}}$. This parameterization requires $n > 2p \cdot \frac{c_{b,\mathsf{asic}}}{c_{r,\mathsf{asic}}} > 2m \cdot \frac{c_{b,\mathsf{asic}}}{c_{r,\mathsf{asic}}}$, which means the capacity requirement of scrypt should be a few hundred times larger than an adversary's conceivable cache size.

5.2 Bit-Reversal Graphs

A bit-reversal graph is a sandwich graph where n is a power of 2 and the cross edges (v_i, u_j) follow the bit-reversal permutation, namely, the binary representation of j reverses the bits of the binary representation of i. Figure 2 is a bit-reversal graph with $n = 8$. Catena-BRG [35] is based on bit-reversal graphs.

Black pebbling complexity. For a black pebble game, Lenguaer and Tarjan [38] showed an asymptotically tight space-time trade-off $ST = \Theta(n^2)$ for bit-reversal graphs.

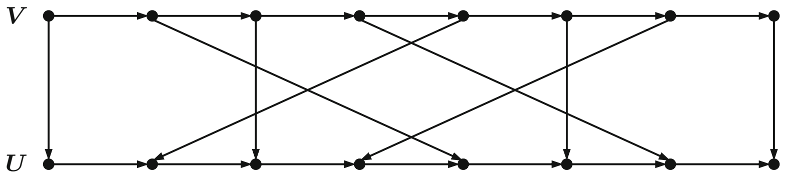

Fig. 2. A bit-reversal graph with $n = 8$.

Red-Blue pebbling complexity. For a red-blue pebble game, the default strategy is the same as the one for scrypt in Sect. 5.1 The total cost of this strategy is $(c_r + c_b)n + (c_b + 2c_r)n = (2c_b + 3c_r)n$. We now show a lower bound on the red-blue pebbling complexity for bit-reversal graphs. The techniques are similar to Lenguaer and Tarjan [38].

Theorem 2. *Let G be a bit-reversal graph with $2n$ vertices, and m be the number of red pebbles available. If $n > 2mc_b/c_r$, then the red-blue pebbling cost $\mathsf{ec}(G, m, c_b, c_r)$ is lower bounded by $(c_b + c_r)n(1 - \frac{2(m+1)c_b}{nc_r})$.*

Proof. Suppose a sequence $\mathbf{P}$ pebbles u_{n-1} of a bit-reversal graph starting from an empty initial configuration. Let m' be the largest power of 2 satisfying $m' < nc_r/c_b$. We have $m' \geq nc_r/(2c_b) > m$.

Let the output path be divided into n/m' intervals of length m' each. Denote the j-th interval I_j, $j = 1, 2, \cdots, n/m'$. I_j contains vertices

$u_{(j-1)m'}, u_{(j-1)m'+1}, \ldots, u_{jm'-1}$. The first time these intervals are pebbled must be in topological order, so $\mathbf{P}$ can be divided into n/m' subsequences $(\mathbf{P}_1, \mathbf{P}_2, \cdots, \mathbf{P}_{n/m'})$ such that all vertices in I_j are pebbled for the first time by $\mathbf{P}_j$. The red blue pebbling costs of subsequences are additive, so we can consider each $\mathbf{P}_j$ separately.

Suppose $\mathbf{P}_j$ uses b blue moves. For any I_j, $1 \le j \le n/m'$, let $v_{j_1}, v_{j_2}, \ldots, v_{j_{m'}}$ be the immediate predecessors on the input path. Note that these immediate predecessors are n/m' edges apart from each other due to the bit-reversal property. $\mathbf{P}_j$ must place red pebbles on all these immediate predecessors at some point during $\mathbf{P}_j$. An immediate predecessor v may get its red pebble in one of the following three ways below:

1. v has a red pebble on it at the beginning of $\mathbf{P}_j$.
2. v has a "close" ancestor (can be itself) that gets a red pebble through a blue move, where being "close" means being less than n/m' edges away. $\mathbf{P}_j$ can then place a red pebble on v using less than n/m' red moves utilizing its "close" ancestor.
3. v gets its red pebble through a red move *and* has no "close" ancestor that gets a red pebble through a blue move. To place a red pebble on v, $\mathbf{P}_j$ must use at least n/m' red moves (except for one v that may be "close" to the source vertex v_0).

The first category accounts for at most m immediate predecessors due to the cache size limit m. The second category accounts for at most b immediate predecessors since each uses a blue move. If $b + m < m' - 1$, then $\mathbf{P}_j$ must use at least n/m' red moves for each immediate predecessor in the third category. Under the conditions in the theorem, the cost of n/m' red moves is greater than a blue move since $c_r n/m' > c_b$. Thus, the best strategy is to use blue moves over red moves for vertices on the input path whenever possible. Therefore,

$$\mathsf{ec}(\mathbf{P}_j) \ge c_r m' + c_b(m' - m - 1) > (c_b + c_r)(m' - m - 1)$$
$$\mathsf{ec}(\mathbf{P}) = \Sigma_{j=1}^{n/m'} \mathsf{ec}(\mathbf{P}_j) > (c_b + c_r)n\left(1 - \frac{2(m+1)c_b}{nc_r}\right). \qquad \square$$

When n is sufficiently large, a bit-reversal graph is bandwidth hard. Its red-blue pebbling complexity has a lower bound close to $(c_b + c_r)n$. An ASIC's energy advantage is similar to that of scrypt, $A_{\mathsf{ec}} \approx \frac{2c_{b,\mathsf{cpu}} + 3c_{r,\mathsf{cpu}}}{c_{b,\mathsf{asic}} + c_{r,\mathsf{asic}}}$ and $A_{\mathsf{ec}} \approx 18$ with parameters in Table 1. The capacity requirement on bit-reversal graphs to remain bandwidth hard is also similar to the requirement for scrypt.

5.3 Stacked Expanders

An (n, α, β) bipartite expander $(0 < \alpha < \beta < 1)$ is a directed bipartite graph with n sources and n sinks such that any subset of αn sinks are connected to at least βn sources. Prior work has shown that bipartite expanders for any $0 < \alpha < \beta < 1$ exist given sufficiently many edges. For example, Pinsker's construction [48] simply connects each sink to d independent sources. It yields an

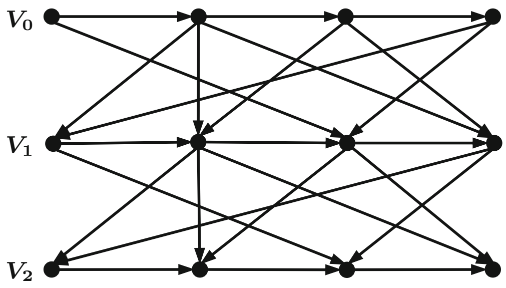

Fig. 3. A stacked random sandwich graph with $n = 4$, $k = 2$ and $d = 2$.

(n, α, β) bipartite expander for sufficiently large n with overwhelming probability [25] if

$$d > \frac{\mathrm{H_b}(\alpha) + \mathrm{H_b}(\beta)}{-\alpha \log_2 \beta}$$

where $\mathrm{H_b}(\alpha) = -\alpha \log_2 \alpha - (1 - \alpha) \log_2(1 - \alpha)$.

An (n, k, α, β) stacked expander graph is constructed by stacking k bipartite expanders back to back. It has $n(k + 1)$ vertices, partitioned into $k + 1$ sets each of size n, $V = \{V_0, V_1, V_2, \cdots, V_k\}$ with all edges are directed from V_{i-1} to V_i ($i \in [1..k]$). V_{i-1} and V_i plus all edges between them form an (n, α, β) bipartite expander $\forall i \in [1..k]$. The bipartite expanders at different layers can but do not have to be the same. Its maximum in-degree is the same as the underlying (n, α, β) bipartite expanders.

In the Balloon hashing algorithm, the vertices are furthered chained sequentially, i.e., there exist edges $(v_{i,j}, v_{i,j+1})$ for each $0 \leq i \leq k, 0 \leq j \leq n-2$ as well as an edge $(v_{i,n-1}, v_{i+1,0})$ for each $0 \leq i \leq k$. In other words, Balloon hashing uses a stacked random sandwich graph in which each vertex has $d > 1$ predecessors from the previous layer. Figure 3 is a stacked random sandwich graph with $n = 4$, $k = 2$ and $d = 2$. In the figure, two consecutive layers form a $(4, 4, \frac{1}{4}, \frac{1}{2})$ expander.

A large in-degree can be problematic for the pebbling abstraction since the random oracle in graph labeling cannot be based on a Merkle-Damgård construction. In the case of stacked expanders, we can apply a transformation to make the in-degree to be 2: we simply replace each d-to-1 connection with a binary tree where the d predecessors are at the leaf level, the successor is the root and each edge in the tree points from a child to its parent. This transformation preserves the expanding property between layers and increases the cost of a red move by a factor of 2 at most (the number of edges in a binary tree is at most twice the number of its leaves).

We remark that the latest version of the Balloon hash paper [25] analyzes random sandwich graphs using a new "well-spread" property rather than the expanding property, in an attempt to tighten the required in-degree. We may be able to adopt their new framework to analyze bandwidth hardness, but we leave it to future work.

Black pebbling complexity. Black pebble games on stacked expanders have been well studied. Obviously, simply pebbling each expander in order and removing pebbles as they are no longer needed results in a sequence $\mathbf{P}$ that uses $2n$ space and $n(k+1)$ moves. An exponentially sharp space-time trade-off in black pebble games is shown by Paul and Tarjan [44] and further strengthened by Ren and Devadas [49]. The result says that to pebble any subset of αn initially unpebbled sinks of G requires either at least $(\beta - 2\alpha)n$ pebbles or at least 2^k moves.

Red-blue pebbling complexity. We now consider red-blue pebble games on stacked expanders. An honest user would simply pebble each expander in order in a straightforward way. First, for each vertex v in the source layer V_0, the honest user places a red pebble on v and then immediately replaces it with a blue pebble. Then, for each vertex $v \in V_1$, the honest user places red pebbles on its d predecessors through blue moves, pebbles v using a red move, replacing the red pebble with a blue pebble, and lastly removing all red pebbles. The cost to pebble each source vertex is $(c_b + c_r)$ and the cost to pebble each non-source vertex is $c_b(d+1) + c_r d$. The total cost to pebble the entire graph is therefore $\approx nkd(c_b + c_r)$.

Following the proof of the sharp space-time trade-off in a black pebble game [44,49], we can similarly derive a sharp trade-off between red and blue moves in a red-blue pebble game. It will then lead to a lower bound on red-blue pebbling cost for stacked expander graph G.

Theorem 3. *Let G be an (n, k, α, β) stacked expander. In a red-blue pebble game, if a sequence $\mathbf{P}$ pebbles any subset of αn sinks of G through red moves, using at most m red pebbles (plus an arbitrary number of blue pebbles) and at most $(\beta - 2\alpha)n - m$ blue moves, then $\mathbf{P}$ must use at least $2^k \alpha n$ red moves.*

Informally, if there is a strategy that pebbles any subset of αn vertices using at most m red *pebbles* and at most b blue *moves*, it implies a strategy that pebbles those αn vertices using at most $m + b$ black pebbles. The reason is that while there may be arbitrarily many blue pebbles, at most b blue pebbles can be utilized since there are at most b blue moves. Therefore, either $m + b \geq (\beta - 2\alpha)n$ or an exponential number of red moves are needed. Below is a rigorous proof.

Proof. The proof is similar to the inductive proof for the black pebble game trade-off [44,49]. For the base case $k = 0$, an $(n, 0, \alpha, \beta)$ stacked expander is simply a collection of n isolated vertices with no edges. The theorem is trivially true since the αn are pebbled through red moves.

Now we show the inductive step for $k \geq 1$ assuming the theorem holds for $k - 1$. The αn sinks in V_k that are pebbled through red moves collectively are connected to at least βn predecessors in V_{k-1} due to the (n, α, β) expander property. Each of these βn vertices in V_{k-1} must have a red pebble on it at some point to facilitate the red moves on the αn sinks. These βn vertices may get their red pebbles in one of the three ways below. Up to m of them may initially have red pebbles on them. Up to $(\beta - 2\alpha)n - m$ of them may initially have blue pebbles on them get red pebbles through blue moves. The remaining $2\alpha n$ of them must get their red pebbles through red moves. These $2\alpha n$ vertices in V_{k-1} are sinks of an $(n, k-1, \alpha, \beta)$ stacked expander. Divide them into two groups of αn each in the order they get red pebbles in $\mathbf{P}$ for the first time. $\mathbf{P}$ can be then divided into two parts $\mathbf{P} = (\mathbf{P}_1, \mathbf{P}_2)$ where $\mathbf{P}_1$ places red pebbles on the first group ($\mathbf{P}_1$ does not place red pebbles on any vertices in the second group) and $\mathbf{P}_2$ places red pebbles on the second group. Both $\mathbf{P}_1$ and $\mathbf{P}_2$ use no more than m red pebbles and $(\beta - 2\alpha)n - m$ blue moves. Due to the inductive hypothesis, both use at least $2^{k-1}\alpha n$ red moves. Therefore, $\mathbf{P}$ uses at least $2^k \alpha n$ red moves. □

Theorem 4. *Let G be an (n, k, α, β) stacked expander with in-degree d. Its red-blue pebbling complexity* $\mathsf{ec}(G, m, c_b, c_r)$ *is lower bounded by* $(c_b+c_r)\cdot((\beta-2\alpha)n-m)\cdot(k - \lceil \log_2(c_b/dc_r)\rceil)/\alpha$.

Proof. With the chain edges, if a sequence starts from an empty configuration, then it must pebble vertices in G in topological order. For simplicity, let us assume each layer of n vertices can be divided into an integer number of groups of size αn each (i.e., αn divides n). Now we can break up the sequence into $(k+1)n/\alpha n$ sub-sequences; each one pebbles the next consecutive αn vertices *for the first time*. Since the red-blue pebbling costs from multiple sub-sequences are additive, we analyze and lower bound them independently.

Consider a sub-sequence $\mathbf{P}'$ that pebbles αn vertices in V_i for the first time. Theorem 3 shows a trade-off on the usage of red versus blue moves. We note that Theorem 3 can be generalized. If $\mathbf{P}'$ uses at most m red pebbles (plus an arbitrary number of blue pebbles) and at most $(\beta - q\alpha)n - m$ blue moves, then $\mathbf{P}'$ must use at least $q^i \alpha n$ red moves. For a proof, simply notice that there will be $q\alpha n$ vertices in V_{i-1} that need to get their red pebbles through red moves. The q^i factor follows from a similar induction. This means $\mathbf{P}'$ must choose one of the following options:

- use at least $(\beta - 2\alpha)n - m$ blue moves, plus αn red moves;
- use less than $(\beta - 2\alpha)n - m$ but at least $(\beta - 3\alpha)n - m$ blue moves, plus at least $2^i \alpha n$ red moves;
- use less than $(\beta - 3\alpha)n - m$ but at least $(\beta - 4\alpha)n - m$ blue moves, plus at least $3^i \alpha n$ red moves;
- $\cdots$

Comparing these options, we see that in order to save αn blue moves, $\mathbf{P}'$ needs to compensate with $(2^i - 1)\alpha n$ more red moves. For $i > \lceil \log_2(c_b/dc_r)\rceil$, blue moves are not worth saving because αn blue moves cost $c_b \alpha n$ which is less than the cost of $2^i - 1$ red moves. To save $(q+1)\alpha n$ blue moves, $\mathbf{P}'$ needs to compensate with $(q^i - 1)\alpha n$ more red moves, which is even less economical. This means, for layers relatively deep, the best strategy is to use blue moves whenever possible. The cost of the first option is $\mathsf{ec}(\mathbf{P}') \geq c_b((\beta-2\alpha)n-m)+c_r d\alpha n > (c_b+c_r)((\beta-2\alpha)n-m)$. The latter inequality is due to $d\alpha > \beta - 2\alpha$, which is easy to check from the requirement on d for expanders. Lastly, there are $(k - \lceil \log_2(c_b/dc_r)\rceil)$ layers V_i

satisfying $i > \lceil \log_2(c_b/dc_r) \rceil$ and each contains $1/\alpha$ vertex groups of size αn. The bound in the theorem follows. $\square$

For a stacked expander graph to be bandwidth hard, we only need n and k to be a constant factor larger than $m/(\beta - 2\alpha)$ and $\lceil \log_2(c_b/dc_r) \rceil$, respectively, which can be much less space and time from honest users compared to scrypt and bit-reversal graphs under some parameters. When n and k are sufficiently large, an ASIC's advantage $A_{\mathsf{ec}} \approx \frac{c_{b,\mathsf{cpu}} + c_{r,\mathsf{cpu}}}{c_{b,\mathsf{asic}} + c_{r,\mathsf{asic}}} \cdot \frac{d\alpha}{\beta - 2\alpha} = \overline{A_{\mathsf{ec}}} \cdot \frac{d\alpha}{\beta - 2\alpha}$. For an example design point, if we can choose $\alpha = 0.01$ and $\beta = 0.05$, we have $d = 9$, $d\alpha/(\beta - 2\alpha) = 3$ and $A_{\mathsf{ec}} \approx 20$.

5.4 Stacked Butterfly Graphs Are Not Bandwidth Hard

In this section, we demonstrate that a capacity hard function in the sequential model may not be bandwidth hard using a stacked double butterfly graph as a counterexample. A double butterfly graph consists of two fast Fourier transform graphs back to back. It has n sources, n sinks and $2n \log_2 n$ vertices in total for some n that is a power of 2. The intermediate vertices form two smaller double butterfly graphs each with $n/2$ sources and $n/2$ sinks. Figure 4 shows an example double butterfly graph with $n = 8$. A stacked double butterfly graph further stacks copies of double butterfly graphs back to back (not shown in the figure).

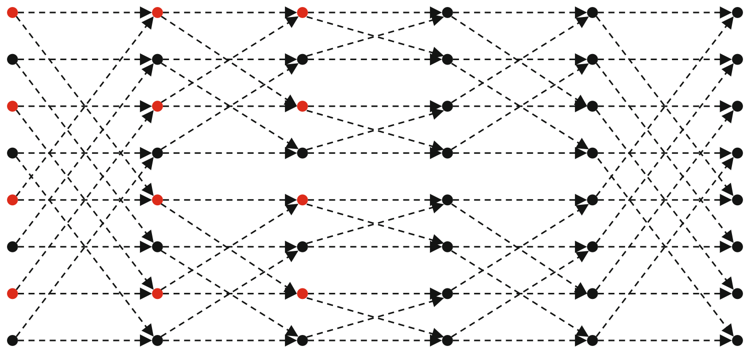

Fig. 4. A double butterfly graph with $n = 8$ sources and sinks. Vertices marked in red have locality assuming a cache size of $m = 4$. (Color figure online)

A double butterfly graph is a superconcentrator [26], and it has been shown that stacked superconcentrators have an exponentially sharp space-time trade-off in sequential pebble games [38]. However, a stacked double butterfly graph is not bandwidth hard due to locality in its memory access pattern. One can fetch a batch of operands into the cache and perform a lot of computation with them before swapping in another batch of operands. For example, in Fig. 4, one can pebble the red vertices layer by layer without relying on other vertices, since the red vertices only have incoming edges from themselves. If equipped with a cache of size m (assume m is a power of 2 for simplicity), we can adopt the following pebbling strategy to save blue moves without sacrificing red moves. We first place red pebbles on m vertices in the same layer that are n/m away from each other, possibly through blue moves. We then use red moves to proceed $\log_2 m$ layers horizontally, placing red pebbles on the m vertices in the same horizontal positions. For a stacked double butterfly with N vertices in total, this strategy uses N red moves and only $N/\log_2 m$ blue moves. Its cost is therefore $(2c_r + c_b/\log_2 m)N$. As demonstrated in Sect. 4.1, $c_{r,\mathsf{cpu}}$ is larger or at least comparable to $c_{b,\mathsf{cpu}}$ while $c_{r,\mathsf{asic}} \ll c_{b,\mathsf{asic}}$. Therefore, the red-blue pebbling of a stacked double butterfly graph costs more than $2c_{r,\mathsf{cpu}}N$ on CPUs and roughly $c_{b,\mathsf{asic}}N/\log_2 m$ on ASICs. This results in an advantage proportional to $\log_2 m$.

Stacked double butterfly graphs are used by the capacity hard function Catena-DBG [35] and the capacity hard proof of work by Ateniese et al. [18]. We note that Catena-DBG designers further add chain edges within each layer [35]. These chain edges will prevent our proposed pebbling strategy, so Catena-DBG may not suffer from the $\log_2 m$ ASIC energy advantage. Our goal here is to show that capacity hard functions in the sequential model are not necessarily bandwidth hard, and a stacked double butterfly graph without chain edges serves as an example. We also remark that since stacked double butterfly graphs are not capacity hard under parallel attacks, it remains unclear whether parallel capacity hardness implies bandwidth hardness. But since capacity and bandwidth are quite different metrics, we currently do not expect that to be the case. (Bandwidth hardness certainly does not imply parallel capacity hardness with Catena-BRG and Balloon being counterexamples.)

6 Discussion

6.1 The Role of Memory

It is not a coincidence that memory plays a central role in both the area and the energy approach, and this point may be worth some further justification. As mentioned, CPUs are at a huge disadvantage over ASICs in computation because CPUs are general purpose while ASICs are specifically optimized for a certain task. Memory does not suffer from this problem because memory is, for the most part, intrinsically general purpose. Memory's job is to store bits and deliver them to the compute units regardless of what the computational task is. New technologies like 3D-stacked memory [24], high speed serial [2] and various types of non-volatile memory do not undermine this argument: they are also general purpose and will be adopted by both CPUs and ASICs when they become successful.

6.2 Capacity Hardness and Energy?

Here a reader may wonder whether we can make an energy argument for capacity hard functions. Specifically, one may argue that holding a large capacity of data in memory also costs energy, and it must be similar for ASICs and CPUs due to the general purpose nature of memory. The problem is that the energy cost of holding data in memory depends on the underlying memory technology, and can be extremely small. We call the power spent on holding data *idle power*, and the power spent on transferring data *busy power*. Volatile memory like DRAM needs to periodically refresh data and thus has a noticeable idle power consumption. For non-volatile memory/storage, idle power is negligible or even strictly 0, independent of memory capacity [54]. Think about hard disks in one's garage — they consume no energy no matter how much data they are holding. The energy argument for capacity hardness breaks down for non-volatile memory/storage. Energy consumption in non-volatile memory/storage only occurs when data transfer happens, which is exactly what our model assumes.

In fact, even with volatile memory like DRAM, the energy model cannot be solely based on memory capacity. While DRAM idle power is indeed proportional to memory capacity, idle power will never be the dominant part in a reasonable system. Section 6.3 further discusses this issue.

6.3 Implications of Parallel Attacks

Parallel attacks and area. Percival [46] defines memory hard functions to be functions that (1) can be computed using T_0 time and $S_0 = O(T_0^{1-\epsilon})$ space, and (2) cannot be computed using T time and S space where $ST = O(T_0^{2-\epsilon})$. The ST lower bound at the first glance makes intuitive sense as it lower bounds the AT product assuming that memory rather than the compute unit dominates chip area. However, concerns have been raised about the above reasoning [10,22]. Because a ST lower bound allows space-time trade-off, a chip designer can reduce the amount of memory by a factor of q, and then use q compute units in parallel to keep the running time at T_0. If q is not too large, chip area may still be dominated by memory, so in theory this parallel architecture reduces the AT product by roughly a factor of q. To address this issue, subsequent proposals introduce stronger notions of capacity hardness that, for example, require a linear space lower bound (in a computational sense) $S = cS_0$ [18,25,35]. But it is later uncovered that parallel architectures can asymptotically decrease the amortized AT product of these constructions as well, and even stronger, the amortized AT product of any data independent functions [10].

However, we would like to note that the above parallel attacks adopt an oversimplified hardware model [10,12,14,15,22]: most of them assume unlimited bandwidth for free. In practice, memory bandwidth is a scarce resource and is *the* major bottleneck in parallel computing, widely known as the "memory wall" [17]. Increasing memory bandwidth would inevitably in turn increase chip area and the more fundamental metric manufacturing cost. Only one paper presents simulation results with concrete bandwidth requirements [11]. We laud this effort, but unfortunately, the paper incorrectly chooses energy as the metric, as we explain below. The area model in those attacks [10] looks reasonable, though the memory bandwidth they assume is still too high. It would improve our understanding on this issue if the authors provide simulation results with area as the metric and for a wide range of bandwidth values from GB/s to TB/s.

Parallel attacks and energy. We adopt a sequential model for bandwidth hard functions in which parallelism does not help by definition. We believe this model is reasonable because, to first-order effects, transferring data sequentially or in parallel should consume roughly the same amount of energy. However, some recent works [10,12,15] conclude that parallel attacks will have an asymptotic energy gain for any data independent function, bit-reversal graphs and stacked expanders included in particular, which is the exact opposite of our conclusion. Their conclusions result from a flawed energy model. They assume memory's idle power is proportional to its capacity, which is reasonable assuming volatile memory like DRAM. The flaw is that they explicitly assume that memory idle power keeps increasing with memory capacity to the extent that it eventually dwarfs all other power consumption. On a closer look, the energy advantage they obtain under this model is not due to a parallel ASIC architecture having superb energy efficiency, but rather because the sequential baseline has absurdly high memory idle energy cost (i.e., energy cost for holding data). Under their concrete parameterization [11], if we hash 1 GB of data using one CPU core, the memory idle power/energy will be $5000\times$ greater than all other power/energy cost combined! The mistake in their concrete parameterization is that they incorrectly cite an estimated conversion rate for area density in a prior work [22] as a conversion rate for energy cost, which leads to an overestimate of memory idle power/energy by at least $100,000\times$. But if only the constant is off, asymptotically speaking, isn't it true that as DRAM capacity increases, eventually memory idle power/energy will dwarf other components? The answer is yes, but its implication is rather uninteresting and not concerning. It tells us that a computer with a single CPU core and Terabytes of DRAM will have terrible energy efficiency because it spends too much energy refreshing DRAM. Obviously, no manufacturer will produce and no user will buy such a computer — long before reaching this design point, manufacturers will switch to non-volatile memory or simply stop adding DRAM capacity.

7 Conclusion

ASIC resistance requires both theoretical advancement and accurate hardware understanding. With this work, we would like call attention to arguably the most important aspect of ASIC resistance: energy efficiency. We illustrate that the popular memory (capacity) hardness notion does not capture energy efficiency, and indeed a capacity hard function may not achieve energy fairness. We propose the notion of bandwidth hardness to achieve energy fairness between ASICs and CPUs. We analyze candidate constructions and show that scrypt, Catena-BRG and Balloon hashing provide good energy efficiency fairness with suitable parameters.

We conclude the paper with a summary of provable security of different constructions under different thread models. (1) If memory access pattern leakage is not a concern, then scrypt is a good option, since it enjoys capacity hardness under parallel attacks as well as bandwidth hardness (for which parallel attacks do not help). (2) If we assume the adversary has limited parallelism, then Balloon hash is a good choice since it achieves sequential capacity hardness and bandwidth hardness. (3) In some scenarios (e.g., Bitcoin mining), it may be argued that energy advantage resistance alone is sufficient to thwart ASIC attackers, in which case data-independent bandwidth hard functions (e.g., Catena-BRG and Balloon) can be used despite parallel attacks on their area resistance. (4) If both area and energy resistance are required, memory access pattern must be data-independent and additionally the adversary has extremely high parallelism, then we know of no good candidates. In this situation, area resistance alone must suffer poly-logarithmic loss. Furthermore, good parallel capacity hard constructions to date are highly complex and we have not been able to analyze their bandwidth behaviors. Lastly, we mention that in the first three models, a possible alternative is Argon2 [21], the winner of the Password Hashing Competition. We have not been able to analyze the bandwidth hardness of Argon2, and it remains interesting future work.

Acknowledgements. The authors are grateful to Krzysztof Pietrzak, Joël Alwen and Jeremiah Blocki for valuable discussions.

References

1. Antminer S9 - Bitmain. https://shop.bitmain.com/market.htm?name=antminer_s9_asic_bitcoin_miner. Accessed 04 Feb 2017
2. High Speed Serial - Xilinx. https://www.xilinx.com/products/technology/high-speed-serial.html. Accessed 04 Feb 2017
3. Intel advanced encryption standard instructions (AES-NI). https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni. Accessed 04 Feb 2017
4. Intel power gadget. https://software.intel.com/en-us/articles/intel-power-gadget-20. Accessed 04 Feb 2017
5. Litecoin. https://litecoin.org/
6. Zoom Hash Scrypt ASIC. http://zoomhash.com/collections/asics. Accessed 20 May 2016
7. Intel SHA extensions (2013). https://software.intel.com/en-us/articles/intel-sha-extensions. Accessed 04 Feb 2017
8. Abadi, M., Burrows, M., Manasse, M., Wobber, T.: Moderately hard, memory-bound functions. ACM Trans. Internet Technol. **5**(2), 299–327 (2005)
9. Almeida, L.C., Andrade, E.R., Barreto, P.S.L.M., Simplicio Jr., M.A.: Lyra: password-based key derivation with tunable memory and processing costs. J. Cryptogr. Eng. **4**(2), 75–89 (2014)
10. Alwen, J., Blocki, J.: Efficiently computing data-independent memory-hard functions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 241–271. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53008-5_9
11. Alwen, J., Blocki, J.: Towards practical attacks on Argon2i and balloon hashing. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 142–157. IEEE (2017)
12. Alwen, J., Blocki, J., Pietrzak, K.: Depth-robust graphs and their cumulative memory complexity. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 3–32. Springer, Cham (2017). doi:10.1007/978-3-319-56617-7_1

13. Alwen, J., Chen, B., Pietrzak, K., Reyzin, L., Tessaro, S.: `Scrypt` is maximally memory-hard. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 33–62. Springer, Cham (2017). doi:10.1007/978-3-319-56617-7_2
14. Alwen, J., Gazi, P., Kamath, C., Klein, K., Osang, G., Pietrzak, K., Reyzin, L., Rolınek, M., Rybár, M.: On the memory-hardness of data-independent password-hashing functions. Cryptology ePrint Archive, Report 2016/783 (2016)
15. Alwen, J., Serbinenko, V.: High parallel complexity graphs and memory-hard functions. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, pp. 595–603. ACM (2015)
16. Amrutur, B., Horowitz, M.: Speed and power scaling of SRAM's. IEEE J. Solid-State Circ. **35**(2), 175–185 (2000)
17. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W. et al.: The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley (2006)
18. Ateniese, G., Bonacina, I., Faonio, A., Galesi, N.: Proofs of space: when space is of the essence. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 538–557. Springer, Cham (2014). doi:10.1007/978-3-319-10879-7_31
19. Back, A.: Hashcash-a denial of service counter-measure (2002)
20. Belady, L.A.: A study of replacement algorithms for a virtual-storage computer. IBM Syst. J. **5**(2), 78–101 (1966)
21. Biryukov, A., Dinu, D., Khovratovich, D.: Fast and tradeoff-resilient memory-hard functions for cryptocurrencies and password hashing (2015)
22. Biryukov, A., Khovratovich, D.: Tradeoff cryptanalysis of memory-hard functions. Cryptology ePrint Archive, Report 2015/227 (2015)
23. Biryukov, A., Khovratovich, D.: Equihash: asymmetric proof-of-work based on the generalized birthday problem. In: NDSS (2016)
24. Black, B., Annavaram, M., Brekelbaum, N., DeVale, J., Jiang, L., Loh, G.H., McCaule, D., Morrow, P., Nelson, D.W., Pantuso, D. et al.: Die stacking (3D) microarchitecture. In: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 469–479. IEEE Computer Society (2006)
25. Boneh, D., Corrigan-Gibbs, H., Schechter, S.: Balloon hashing: a memory-hard function providing provable protection against sequential attacks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 220–248. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53887-6_8
26. Bradley, W.F.: Superconcentration on a pair of butterflies. CoRR abs/1401.7263 (2014)
27. Cook, S.A.: An observation on time-storage trade off. In: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing, pp. 29–33. ACM (1973)
28. Damgård, I.B.: A design principle for hash functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, New York (1990). doi:10.1007/0-387-34805-0_39
29. Dwork, C., Goldberg, A., Naor, M.: On memory-bound functions for fighting spam. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 426–444. Springer, Heidelberg (2003). doi:10.1007/978-3-540-45146-4_25
30. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993). doi:10.1007/3-540-48071-4_10

31. Dwork, C., Naor, M., Wee, H.: Pebbling and proofs of work. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 37–54. Springer, Heidelberg (2005). doi:10.1007/11535218_3
32. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 585–605. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48000-7_29
33. Dziembowski, S., Kazana, T., Wichs, D.: One-time computable self-erasing functions. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 125–143. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19571-6_9
34. Forler, C., List, E., Lucks, S., Wenzel, J.: Overview of the candidates for the password hashing competition. In: Mjølsnes, S.F. (ed.) PASSWORDS 2014. LNCS, vol. 9393, pp. 3–18. Springer, Cham (2015). doi:10.1007/978-3-319-24192-0_1
35. Forler, C., Lucks, S., Wenzel, J.: Catena : a memory-consuming password-scrambling framework. Cryptology ePrint Archive, Report 2013/525 (2013)
36. Hopcroft, J., Paul, W., Valiant, L.: On time versus space and related problems. In: 16th Annual Symposium on Foundations of Computer Science, pp. 57–64. IEEE (1975)
37. Horowitz, M.: Computing's energy problem (and what we can do about it). In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 10–14. IEEE (2014)
38. Lengauer, T., Tarjan, R.E.: Asymptotically tight bounds on time-space trade-offs in a pebble game. J. ACM **29**(4), 1087–1130 (1982)
39. Lerner, S.D.: Strict memory hard hashing functions (preliminary v0. 3, 01–19-14)
40. Mahmoody, M., Moran, T., Vadhan, S.: Publicly verifiable proofs of sequential work. In: Proceedings of the 4th Conference on Innovations in Theoretical Computer Science, pp. 373–388. ACM (2013)
41. Mead, C.A., Rem, M.: Cost and performance of vlsi computing structures. IEEE Trans. Electron Devices **26**(4), 533–540 (1979)
42. Merkle, R.C.: One way hash functions and DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, New York (1990). doi:10.1007/0-387-34805-0_40
43. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
44. Paul, W.J., Tarjan, R.E.: Time-space trade-offs in a pebble game. Acta Inf. **10**(2), 111–115 (1978)
45. Pedram, A., Richardson, S., Galal, S., Kvatinsky, S., Horowitz, M.: Dark memory and accelerator-rich system optimization in the dark silicon era. IEEE Des. Test **34**, 39–50 (2016)
46. Percival, C.: Stronger key derivation via sequential memory-hard functions (2009)
47. Peslyak, A.: yescrypt - a password hashing competition submission (2014). https://password-hashing.net/submissions/specs/yescrypt-v2.pdf. Accessed Aug 2016
48. Pinsker, M.S.: On the complexity of a concentrator. In: 7th International Telegraffic Conference, vol. 4 (1973)
49. Ren, L., Devadas, S.: Proof of space from stacked expanders. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9985, pp. 262–285. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53641-4_11
50. Savage, J.E.: Models of Computation. Addison-Wesley, Boston (1998)
51. Sethi, R.: Complete register allocation problems. SIAM J. Comput. **4**(3), 226–248 (1975)
52. Smith, A., Zhang, Y.: Near-linear time, leakage-resilient key evolution schemes from expander graphs. Cryptology ePrint Archive, Report 2013/864 (2013)

53. Tromp, J.: Cuckoo cycle: a memory-hard proof-of-work system (2014)
54. Xue, C.J., Sun, G., Zhang, Y., Yang, J.J., Chen, Y., Li, H.: Emerging non-volatile memories: opportunities and challenges. In: 2011 Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, pp. 325–334. IEEE (2011)