Proof of Space from Stacked Expanders

Ling Ren Srinivas Devadas

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

Abstract. Recently, proof of space (PoS) has been suggested as a more egalitarian alternative to the traditional hash-based proof of work. In PoS, a prover proves to a verifier that it has dedicated some specified amount of space. A closely related notion is memory-hard functions (MHF), functions that require a lot of memory/space to compute. While making promising progress, existing PoS and MHF have several problems. First, there are large gaps between the desired space-hardness and what can be proven. Second, it has been pointed out that PoS and MHF should require a lot of space not just at some point, but throughout the entire computation/protocol; few proposals considered this issue. Third, the two existing PoS constructions are both based on a class of graphs called superconcentrators, which are either hard to construct or add a logarithmic factor overhead to efficiency. In this paper, we construct PoS from stacked expander graphs. Our constructions are simpler, more efficient and have tighter provable space-hardness than prior works. Our results also apply to a recent MHF called Balloon hash. We show Balloon hash has tighter space-hardness than previously believed and consistent space-hardness throughout its computation.

1 Introduction

Proof of work (PoW) has found applications in spam/denial-of-service countermeasures [22,13] and in the famous cryptocurrency Bitcoin [36]. However, the traditional hash-based PoW does have several drawbacks, most notably poor resistance to application-specific integrated circuits (ASIC). ASIC hash units easily offer $\sim 100\times$ speedup and $\sim 10,000\times$ energy efficiency over CPUs. This gives ASIC-equipped adversaries a huge advantage over common desktop/laptop users. Recently, proof of space (PoS) [11,24] has been suggested as a potential alternative to PoW to address this problem. A PoS is a protocol between two parties, a prover and a verifier. Analogous to (but also in contrast to) PoW, the prover generates a cryptographic proof that it has invested a significant amount of memory or disk space (as opposed to computation), and the proof should be easy for the verifier to check. It is believed that if an ASIC has to access a large external memory, its advantage over a CPU will be small, making PoS more egalitarian than PoW.

Somewhat unfortunately, two competing definitions of "proof of space" have been proposed [11,24] with very different security guarantees and applications. Adding to the confusion are other closely related and similar-sounding notions

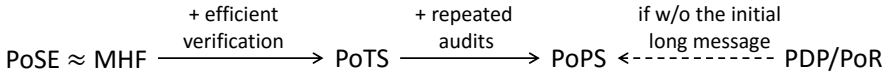

Fig. 1: Relation between PoSE/MHF, PoTS, PoPS and PDP/PoR.

such as memory-hard functions (MHF) [39], proof of secure erasure (PoSE) [40], provable data possession (PDP) [12] and proof of retrievability (PoR) [30]. A first goal of this paper is to clarify the connections and differences between these notions. Section 2 will give detailed comparisons. For now, we give a short summary below and in Figure 1.

As its name suggests, a memory-hard function (MHF) is a function that requires a lot of memory/space to compute. Proof of secure erasure (PoSE) for the most part is equivalent to MHF. Proof of space by Ateniese et al. [11] extends MHF with efficient verification. That is, a verifier only needs a small amount of space and computation to check a prover's claimed space usage. Proof of space by Dziembowski et al. [24] further gives a verifier the ability to repeatedly audit a prover and check if it is still storing a large amount of data. The key difference between the two proofs of space lies in whether the proof is for transient space or persistent space. We shall distinguish these two notions of space and define them separately as *proof of transient space (PoTS)* and *proof of persistent space (PoPS)*. PDP and PoR solve a very different problem: availability check for a user's outsourced storage to an untrusted server. They do force the server to use a lot of persistent space but do not meet the succinctness (short input) requirement since a large amount of data needs to be transferred initially. Since PoPS is the strongest among the four related primitives (MHF, PoSE, PoTS, and PoPS), the end goal of this paper will be a PoPS with improved efficiency and space-hardness. Along the way, our techniques and analysis improve MHF/PoSE and PoTS as well.

Let us return to the requirements for a MHF f. Besides being space-hard, it must take short inputs. This is to rule out trivial solutions that only take long and incompressible input x of size $|x| = N$. Such an f trivially has space-hardness since N space is needed to receive the input, but is rather uninteresting. We do not have many candidate problems that satisfy both requirements. Graph pebbling (also known as pebble games) in the random oracle model is the only candidate we know of so far. Although the random oracle model is not the most satisfactory assumption from a theoretical perspective, it has proven useful in practice and has become standard in this line of research [26,31,11,24,28,8,20]. Following prior work, we also adopt the graph pebbling framework and the random oracle model in this work.

A pebble game is a single-player game on a directed acyclic graph (DAG). The player's goal is to put pebbles on certain vertices. A pebble can be placed on a vertex if it has no predecessor or if all of its predecessors have pebbles on them. Pebbles can be removed from any vertex at any time. The number of pebbles on the graph models the space usage of an algorithm.

Pebble games on certain graphs have been shown to have high space complexity or sharp space-time trade-offs. The most famous ones are stacked superconcentrators [38,32], which have been adopted in MHF [28], PoSE [31] and PoS [11,24]. However, bounds in graph pebbling are often very loose, especially for stacked superconcentrators [38,32]. This translates to large gaps between the desired memory/space-hardness and the provable guarantees in MHF and PoS (Section 1.1). Furthermore, MHFs and PoS need other highly desired properties that have not been studied in graph pebbling before (Section 1.2). The main contribution of this paper is to close these unproven or unstudied gaps while maintaining or even improving efficiency.

We illustrate these problems in the next two subsections using MHF as an example, but the analysis and discussion apply to PoS as well. We use "memory-hard" and "space-hard" interchangeably throughout the paper.

1.1 Gaps in Provable Memory Hardness

The most strict memory-hardness definition for a MHF f is that for any x, $f(x)$ can be efficiently computed using N space, but is impossible to compute using $N-1$ space. Here, "impossible to compute" means the best strategy is to take a random guess in the output space of $f(\cdot)$ (by the random oracle assumption). Achieving this strict notion of memory-hardness is expensive. Aside from the trivial solution that sets input size to $|x| = N$, the best known construction has $O(N^2)$ time complexity for computing f [26]. The quadratic runtime makes this MHF impractical for large space requirements.

All other MHFs/PoSE and PoS in the literature have quasilinear runtime, i.e., $N \cdot \mathsf{polylog} N$, by adopting much more relaxed notions of memory-hardness. One relaxation is to introduce an unproven gap [31,24]. For example, in the PoSE by Karvelas and Kiayias [31], while the best known algorithm to compute f needs N space, it can only be shown that computing f using less than $N/32$ space is impossible. No guarantees can be provided if an adversary uses more than $N/32$ but less than N space.

The other way to relax memory-hardness is to allow space-time trade-offs, and it is usually combined with unproven gaps. Suppose the best known algorithm (to most users) for a MHF takes S space and T time. These proposals hope to claim that any algorithm using $S' = S/q$ space should run for T' time, so that the time penalty T'/T is "reasonable". If the time penalty is linear in q, it corresponds to a lower bound on $S'T' = \Omega(ST)$, as scrypt [39] and Catena-BRG [28] did. Notice that the hidden constant in the bound leaves an unproven gap. Other works require the penalty to be superlinear in q [28,15] or exponential in some security parameter [33,11,20], but the penalty only kicks in when S' is below some threshold, e.g., $N/8$, again leaving a gap.

We believe an exponential penalty is justifiable since it corresponds to the widely used computational security in cryptography. However, an ST lower bound and a large unproven gap are both unsatisfactory. Recall that the motivation of MHF is ASIC-resistance. With an ST bound, an attacker is explicitly allowed to decrease space usage, at the cost of a proportional increase in computation.

Then, an adversary may be able to fit $S/100$ space in an ASIC, and get in return a speedup or energy efficiency gain well over 100.

A large unproven gap leaves open the possibility that an adversary may gain an unfair advantage over honest users, and fairness is vital to applications like voting and cryptocurrency. A more dramatic example is perhaps PoSE [31]. With an unproven gap of 32, a verifier can only be assured that a prover has wiped 1/32 fraction of its storage, which can hardly be considered a "proof of erasure". The authors were well aware of the problem and commented that this gap needs to be very small for a PoSE to be useful [31], yet they were unable to tighten it. Every MHF, PoSE or PoS with quasilinear efficiency so far has a large unproven gap (if it has a provable guarantee at all).

In fact, MHFs have been broken due to the above weaknesses. Scrypt [39], the most popular MHF, proved an asymptotic ST lower bound. But an ST bound does not prevent space-time trade-offs, and the hidden constants in the bounds turned out to be too small to provide meaningful guarantees [16]. As a result, ASICs for scrypt are already commercially available [1]. The lesson is that space-hardness is one of the examples where exact security matters. PoS proposals so far have not received much attention from cryptanalysis, but the loose hidden constants in prior works are equally concerning. Therefore, we will be explicit about every constant in our constructions, and also make best efforts to analyze hidden constants in prior works (in Tables 1, 2 and 3).

1.2 Consistent Memory Hardness

In a recent inspiring paper, Alwen and Serbinenko pointed out an overlooked weakness in all existing MHFs' memory-hardness guarantees [8]. Again, the discussion below applies to PoS. The issue is that in current definitions, even if a MHF f is proven to require N space in the most strict sense, it means N space is needed *at some point* during computation. It is possible that f can be computed by an algorithm that has a short memory-hard phase followed by a long memory-easy phase. In this case, an adversary can carry out the memory-hard phase on a CPU and then offload the memory-easy phase to an ASIC, defeating the supposed ASIC-resistance.

Alwen and Serbinenko argue, and we fully agree, that a good MHF should require a lot of memory not just at some point during its computation, but throughout the majority of its computation. However, we think the solution they presented has limitations. Alwen and Serbinenko suggested lower bounding a MHF's cumulative complexity (CC), the sum of memory usage in all steps of an algorithm [8]. For example, if the algorithm most users adopt takes T time and uses S space at every time step, its CC is ST. If we can lower bound the CC of any algorithm for this MHF to ST, it rules out an algorithm that runs for T time, uses S space for a few steps but very little space at other steps. A CC bound is thus an improved version of an ST bound, and this is also where the problem is. Like an ST bound, CC explicitly allows proportional space-time trade-offs: algorithms that run for qT time and use S/q space for any factor q. Even when combined with a strict space lower bound of S, it still does not

rule out an algorithm that runs for qT time, uses S space for a few steps but S/q space at all other steps. We have discussed why a proportional space-time trade-off or a long memory-easy phase can be harmful, and CC allows both.

Instead, we take a more direct approach to this problem. Recall that our goal is to design a MHF that consistently uses a lot of memory during its computation. So we will simply lower bound the number of time steps during the computation with high space usage. If this lower bound is tight, we say a MHF has consistent memory-hardness.

Another difference between our approach and that of [8] is the computation model. Alwen and Serbinenko assumed their adversaries possess infinite parallel processing power, and admirably proved lower bounds for their construction against such powerful adversaries. But their construction is highly complicated and the bound is very loose. We choose to stay with the sequential model or limited parallelism for two reasons. First, cumulative/consistent memory-hardness and parallelism are two completely independent issues and should not be coupled. Consistent (cumulative) memory-hardness is extremely important in the sequential model. Mixing it with the parallel model gives the wrong impression that it only becomes a problem when an adversary has infinite parallelism. Second, massive parallelism seems unlikely for MHFs in the near future. Even if parallel computation is free in ASICs, to take advantage of it, an adversary also needs proportionally higher memory bandwidth (at least in our construction). Memory bandwidth is a scarce resource and is *the* major bottleneck in parallel computing, widely known as the "memory wall" [10]. It is interesting to study the infinitely parallel model from a theoretical perspective as memory bandwidth may become cheap in the future. But at the time being, it is not worth giving up practical and provably secure solutions in the sequential model.

1.3 Our Results

We construct PoTS and PoPS from stacked expanders. Our constructions are conceptually simpler, more efficient and have tighter space-hardness guarantees than prior works [11,24]. We could base our space-hardness on a classical result by Paul and Tarjan [37], but doing so would result in a large unproven gap. Instead, we carefully improve the result by Paul and Tarjan to make the gap arbitrarily small. We then introduce the notion of consistent memory-hardness and prove that stacked expanders have this property.

These results lead to better space-hardness guarantees for our constructions. For our PoTS, we show that no computationally bounded adversary using γN space can convince a verifier with non-negligible probability, where γ can be made arbitrarily close to 1. The prover also needs close to N space not just at some point in the protocol, but consistently throughout the protocol. In fact, the honest strategy is very close to the theoretical limits up to some tight constants. For PoPS, we show that an adversary using a constant fraction of N persistent space (e.g., $N/3$) will incur a big penalty. It is a bit unsatisfactory that we are unable to further tighten the bound and have to leave a small gap. But our result

still represents a big improvement over the only previous PoPS [24] whose gap is as large as $2 \times 256 \times 25.3 \times \log N$.

Our tight and consistent memory-hardness results can be of independent interest. Independent of our work, Corrigan-Gibbs et al. recently used stacked expanders to build a MHF called Balloon hash [20]. They invoked Paul and Tarjan [37] for space-hardness and left an unproven gap of 8. (Their latest space-hardness proof no longer relies on [37], but the gap remains the same.) Our results show that Balloon hash offers much better space-hardness than previously believed. Our work also positively answers several questions left open by Corrigan-Gibbs et al. [20]: Balloon hash is consistently space-hard, over time and under batching.

2 Related Work

MHF. 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. However, ASIC hash units have made the brute force attack considerably easier. This motivated memory-hard functions (MHF) as better password scramblers. Percival [39] proposed the first MHF, scrypt, as a way to derive keys from passwords. Subsequent works [28,33,3,15,20] continued to study MHFs as key derivation functions, password scramblers, and more recently as proof of work. In the recent Password Hashing Competition [27], the winner Argon2 [15] and three of the four "special recognitions"—Catena [28], Lyra2 [3] and yescrypt [41]—claimed memory-hardness.

The most relevant MHF to our work is Balloon hash [20], which also adopted stacked expanders. We adopt a technique from Balloon hash to improve our space-hardness. Our analysis, in turn, demonstrates better space-hardness for Balloon hash and positively answers several open questions regarding its consistent memory-hardness [20]. We also develop additional techniques to obtain PoS.

Attacking MHF. MHFs have been classified into data-dependent ones (dMHF) and data-independent ones (iMHF), based on whether a MHF's memory access pattern depends on its input [28,20]. Catena and Balloon hash are iMHF, and the rest are dMHF. Some consider dMHFs less secure for password hashing due to cache timing attacks.

Most MHF proposals lack rigorous analysis, and better space-time trade-offs (in the traditional sequential model) have been shown against them [16,20]. The only two exceptions are Catena-DBG and Balloon, both of which use graph pebbling. Alwen and Blocki considered adversaries with infinite parallel processing power, and showed that such a powerful attacker can break any iMHF, including Catena-DBG and Balloon [5,6].

MBF. Prior to memory-hard functions, Dwork et al. [21,23] and Abadi et al. [2] proposed memory-bound functions (MBF). The motivation of MBF is also ASIC-resistance, but the complexity metric there is the number of cache misses. A

MHF may not be memory-bound since its memory accesses may hit in cache most of the time. A MBF has to be somewhat memory-hard to spill from cache, but it may not consume too much memory beyond the cache size. A construction that enjoys both properties should offer even better resistance to ASICs, but we have not seen efforts in this direction.

PoSE. Proof of secure erasure (PoSE) was first studied by Perito and Tsudik [40] as a way to wipe a remote device. Assuming a verifier knows a prover (a remote device) has exactly N space, any protocol that forces the prover to use N space was considered a PoSE [40]. This includes the trivial solution where the verifier sends the prover a large random file of size N, and then asks the prover to send it back. Since this trivial solution is inefficient and uninteresting, for the rest of the paper when we say PoSE, we always mean communication-efficient PoSE [31], where the prover receives a short challenge but needs a lot of space to generate a proof. A reader may have noticed that space-hardness and short input are exactly the same requirements we had earlier for MHFs. Thus, we can think of PoSE as an application of MHFs, with one small caveat in the definition of space-hardness. We have mentioned that a proportional space-time trade-off or a large unproven gap are undesirable for MHFs; for PoSE, they are unacceptable. On the flip side, PoSE does not need consistent space-hardness.

PoTS and memory-hard PoW. Two independent works named their protocols "proofs of space" [11,24]. The key difference is whether the proof is for transient space or persistent space. Ateniese et al. [11] corresponds to a proof of transient space (PoTS). It enables efficient verification of a MHF with $\mathsf{polylog}(N)$ verifier space and time. If we simply drop the efficient verification method and have the verifier redo the prover's work, PoTS reduces to PoSE/MHF.

Two recent proposals Cuckoo Cycle [48] and Equihash [17] aim to achieve exactly the same goal as PoTS, and call themselves memory-hard proof of work. This is also an appropriate term because a prover in PoTS has to invest both space and time, usually N space and $N \cdot \mathsf{polylog}(N)$ computation. Cuckoo Cycle and Equihash are more efficient than Ateniese et al. [11] but do not have security proofs. An attack on Cuckoo Cycle has already been proposed [9].

PoPS. Dziembowski et al. [24] is a proof of persistent space (PoPS). Compared to Ateniese et al. [11], it supports "repeated audits". The protocol has two stages. In the first stage, the prover generates some data of size N, which we call *advice*. The prover is supposed to store the advice persistently throughout the second stage. In the second stage, the verifier can repeatedly audit the prover and check if it is still storing the advice. All messages exchanged between the two parties and the verifier's space/time complexity in both stages should be $\mathsf{polylog}(N)$. If the prover is audited only once, PoPS reduces to PoTS.

It is worth pointing out that an adversary can always discard the advice and rerun setup when audited. To this end, the space-hardness definition is somewhat relaxed (see Section 6 for details). PoPS also attempts to address the other drawback of PoW: high energy cost. It allows an honest prover who faithfully

stores the advice to respond to audits using little computation, hence consuming little dynamic energy. Whether these features are desirable depends heavily on the application. Proof of Space-Time [35] is a recent proposal that resembles PoPS but differs in the above two features. In their protocol, an honest prover needs to access the entire N-sized advice or at least a large fraction to pass each audit. In return, they argue that the penalty they guarantee for a cheating prover is larger.

PDP and PoR. Provable data possession (PDP) [12] and proof of retrievability (PoR) [30] allow a user who outsources data to a server to repeatedly check if the server is still storing his/her data. If a verifier (user) outsources large and incompressible data to a prover (server), PDP and PoR can achieve the space-hardness goal of both PoTS and PoPS. However, transmitting the initial data incurs high communication cost. In this aspect, PoS schemes [11,24] are stronger as they achieve low communication cost. PDP and PoR are stronger in another aspect: they can be applied to arbitrary user data while PoS populates prover/server memory only with random bits. In summary, PDP and PoR solve a different problem and are out of the scope of this paper.

Graph pebbling. Graph pebbling is a powerful tool in computer science, dating back at least to 1970s in studying Turing machines [19,29] and register allocation [46]. More recently, graph pebbling has found applications in various areas of cryptography [23,26,25,47,34,28,31,11,24].

Superconcentrators. The simplest superconcentrator is perhaps the butterfly graph, adopted in MHF/PoSE [28,31] and PoTS [11], but it has a logarithmic factor more vertices and edges than linear superconcentrators or expanders. Linear superconcentrators, adopted in PoPS [24], on the other hand, are hard to construct and recursively use expanders as building blocks [18,44,4,45]. Thus, it is expected that superconcentrator-based MHFs and PoS will be more complicated and less efficient than expander-based ones (under comparable space-hardness).

3 Pebble Games on Stacked Expanders

3.1 Graph Pebbling and Labelling

A pebble game is a single-player game on a directed acyclic graph (DAG) G with a constant maximum in-degree d. A vertex with no incoming edges is called a *source* and a vertex with no outgoing edges is called a *sink*. The player's goal is to put pebbles on certain vertices of G using a sequence of *moves*. In each move, the player can place one pebble and remove an arbitrary number of pebbles (removing pebbles is free in our model). The player's moves can be represented as a sequence of transitions between pebble placement configurations on the graph, $\mathbf{P} = (P_0, P_1, P_2 \cdots, P_T)$. If a pebble exists on a vertex v in a configuration P_i, we say v is *pebbled* in P_i. The starting configuration P_0 does not have to be empty; vertices can be pebbled in P_0. The pebble game rule is as follows: to transition

from P_i to P_{i+1}, the player can *pebble* (i.e., place a pebble on) one vertex v if v is a source or if all predecessors of v are pebbled in P_i, and then *unpebble* (i.e., remove pebbles from) any subset of vertices. 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.

A pebble game is just an abstraction. We need a concrete computational problem to enforce the pebble game rules. Prior work has shown that the graph labelling problem with a random oracle $\mathcal{H}$ implements pebble games. In graph labelling, vertices are numbered, and each vertex v_i is associated with a label $h(v_i) \in \{0,1\}^\lambda$ where λ is the output length of $\mathcal{H}$.

$$h(v_i) = \begin{cases} \mathcal{H}(i, x) & \text{if } v_i \text{ is a source} \\ \mathcal{H}(i, h(u_1), h(u_2), \cdots, h(u_d)) & \text{otherwise, } u_1 \text{ to } u_d \text{ are } v_i\text{'s predecessors} \end{cases}$$

Clearly, any legal pebbling sequence gives a graph labelling algorithm. It has been shown that the converse is also true for PoSE/MHF [23,26,31] and PoTS [11], via a now fairly standard "ex post facto" argument. The equivalence has not been shown for PoPS due to subtle issues [24], but there has been recent progress in this direction [7]. We refer readers to these papers and will not restate their results.

Given the equivalence (by either a proof or a conjecture), we can use metrics of the underlying pebble games to analyze higher-level primitives. Consider a pebble sequence $\mathbf{P} = (P_0, P_1, P_2 \cdots, P_T)$. Let $|P_i|$ be the number of pebbles on the graph in configuration P_i. We define the space complexity of a sequence $S(\mathbf{P}) = \max_i(|P_i|)$, i.e., the maximum number of pebbles on the graph at any step. It is worth noting that space in graph labelling is measured in "label size" λ rather than bits.

We define the time complexity of a sequence $T(\mathbf{P})$ to be the number of transitions in $\mathbf{P}$. $T(\mathbf{P})$ equals the number of random oracle $\mathcal{H}$ calls, because we only allow one new pebble to be placed per move. This corresponds to the sequential model. We can generalize to limited parallelism, say q-way parallelism, by allowing up to q pebble placements per move. But we do not consider infinite parallelism in this paper as discussed in Section 1.2.

For a more accurate timing model in graph labelling, we assume the time to compute a label is proportional to the input length to $\mathcal{H}$, i.e., the in-degree of the vertex. Another way to look at it is that we can transform a graph with maximum in-degree d into a graph with maximum in-degree 2 by turning each vertex into a binary tree of up to d leaves.

To capture consistent space-hardness, we define $M_{S'}(\mathbf{P}) = |\{i : |P_i| \geq S'\}|$, i.e., the number of configurations in $\mathbf{P}$ that contain at least S' pebbles. Consider a pebble game that has a legal sequence $\mathbf{P}$. If there exist some $S' < S(\mathbf{P})$ and $T' < T(\mathbf{P})$, such that any legal sequence $\mathbf{P}'$ for that same pebble game has $M_{S'}(\mathbf{P}') \geq T'$, we say the pebble game is consistently memory-hard. The distance between (S', T') and $(S(\mathbf{P}), T(\mathbf{P}))$ measures the quality of consistent memory-hardness.

3.2 Bipartite Expanders

Now we introduce bipartite expanders, the basic building blocks for our constructions, and review classical results on their efficient randomized constructions.

Definition 1. *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. We adopt the randomized construction by Chung [18]. This construction gives a d-regular bipartite expander, i.e., there are d outgoing edges from each source and d incoming edges to each sink. It simply connects the dn outgoing edges of the sources and the dn incoming edges of the sinks according to a random permutation. Given a permutation Π on $\{0, 1, 2, \cdots, dn-1\}$, if $\Pi(i) = j$, add an edge from source $(i \mod n)$ to sink $(j \mod n)$.

Theorem 1. *Chung's construction yields an (n, α, β) bipartite expander $(0 < \alpha < \beta < 1)$ for sufficiently large n with overwhelming probability if*

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

where $\mathrm{H_b}(\alpha) = -\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha)$ is the binary entropy function.

The theorem has been proven by Bassalygo [14] and Schöning [44], but both proofs were quite involved. We give a proof using a simple counting argument.

Proof. There are $(dn)!$ permutations in total. We analyze how many permutations are "bad", i.e., do not yield an expander. A bad permutation must connect some subset U of αn sinks to a subset V of βn sources. There are $\binom{n}{\alpha n}\binom{n}{\beta n}$ combinations. Within each combination, there are $\binom{d\beta n}{d\alpha n}(d\alpha n)!$ ways to connect U to V. There are $(dn - d\alpha n)!$ ways to connect the rest of edges (those not incident to U). The probability that we hit a bad permutation is

$$\begin{aligned}\Pr(\Pi \text{ is bad}) &= \binom{n}{\alpha n}\binom{n}{\beta n}\binom{d\beta n}{d\alpha n}(d\alpha n)!(dn - d\alpha n)!/(dn)! \\ &= \binom{n}{\alpha n}\binom{n}{\beta n}\binom{d\beta n}{d\alpha n}/\binom{dn}{d\alpha n}\end{aligned}$$

Using Robbins' inequality for Stirling's approximation $\sqrt{2\pi n}(n/e)^n e^{\frac{1}{12n+1}} < n! < \sqrt{2\pi n}(n/e)^n e^{\frac{1}{12n}}$ [43], we have $\log_2 \binom{n}{\alpha n} = n\mathrm{H_b}(\alpha) - \frac{1}{2}\log_2 n + o(1)$. Thus,

$$\log_2 \Pr(\Pi \text{ is bad}) = n[\mathrm{H_b}(\alpha) + \mathrm{H_b}(\beta) + d\beta \mathrm{H_b}(\alpha/\beta) - d\mathrm{H_b}(\alpha)] - \log_2 n + o(1).$$

If $\mathrm{H_b}(\alpha) + \mathrm{H_b}(\beta) + d\beta\mathrm{H_b}(\alpha/\beta) - d\mathrm{H_b}(\alpha) < 0$, or equivalently the bound on d in the theorem statement holds, then $\Pr(\Pi \text{ is bad})$ decreases exponentially as n increases. □

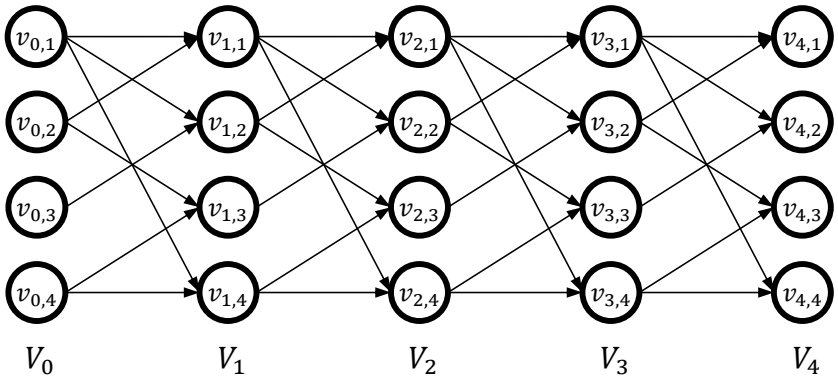

Fig. 2: A stacked bipartite expander $G_{(4,4,\frac{1}{4},\frac{1}{2})}$.

Pinsker [42] used a different randomized construction, which independently selects d predecessors for each sink. Pinsker's construction requires $d > \frac{\mathrm{H_b}(\alpha)+\mathrm{H_b}(\beta)}{-\alpha \log_2 \beta}$ [20], which is a slightly worse bound than Theorem 1. But Pinsker's construction is arguably simpler than Chung's because it only needs a random function as opposed to a random permutation.

3.3 Pebble Games on Stacked Bipartite Expanders

Construct $G_{(n,k,\alpha,\beta)}$ by stacking (n, α, β) bipartite expanders. $G_{(n,k,\alpha,\beta)}$ has $n(k+1)$ vertices, partitioned into $k+1$ sets each of size n, $V = \{V_0, V_1, V_2, \cdots, V_k\}$. All edges in $G_{(n,k,\alpha,\beta)}$ go from V_{i-1} to V_i for some i from 1 to k. For each i from 1 to k, V_{i-1} and V_i plus all edges between them form an (n, α, β) bipartite expander. The bipartite expanders at different layers can but do not have to be the same. $G_{(n,k,\alpha,\beta)}$ has n sources, n sinks, and the same maximum in-degree as the underlying (n, α, β) bipartite expander. Figure 2 is an example of $G_{(4,4,\frac{1}{4},\frac{1}{2})}$ with in-degree 2.

Obviously, simply pebbling each expander in order results in a sequence $\mathbf{P}$ that pebbles $G_{(n,k,\alpha,\beta)}$ using $S(\mathbf{P}) = 2n$ space in $T(\mathbf{P}) = n(k+1)$ moves. Paul and Tarjan [37] showed that $G_{(n,k,\frac{1}{8},\frac{1}{2})}$ has an exponentially sharp space-time trade-off. Generalized to (n, α, β) expanders, their result was the following:

Theorem 2 (Paul and Tarjan [37]). *If* $\mathbf{P}$ *pebbles any subset of* $2\alpha n$ *sinks of* $G_{(n,k,\alpha,\beta)}$*, starting with* $|P_0| \leq \alpha n$ *and using* $S(\mathbf{P}) \leq \alpha n$ *space, then* $T(\mathbf{P}) \geq \lfloor \frac{\beta}{2\alpha} \rfloor^k$.

This theorem forms the foundation of Balloon hash. We could base our PoTS/PoPS protocols on it. However, the space-hardness guarantee we get will be at most $n/4$. We need $\frac{\beta}{2\alpha} \geq 2$ to get an exponential time penalty, so $\alpha n < \beta n/4 < n/4$.

Through a more careful analysis, we show a tighter space-time trade-off for stacked bipartite expanders, which will lead to better space-hardness for

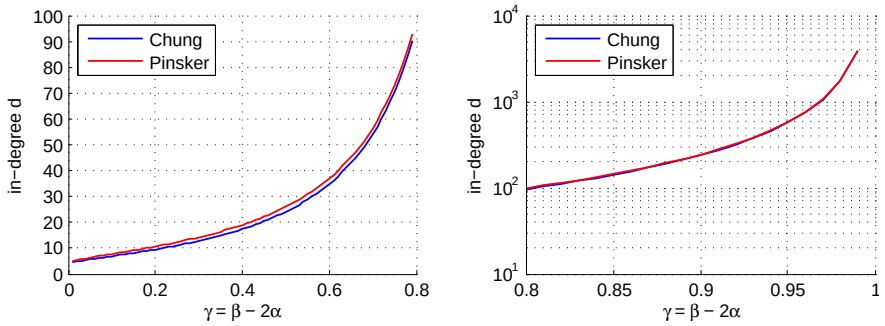

Fig. 3: Minimum in-degree d to achieve a given $\gamma = \beta - 2\alpha$.

our PoTS/PoPS protocols as well as Balloon hash. We improve Theorem 2 by considering only initially unpebbled sinks. Let $\gamma = \beta - 2\alpha > 0$ for the rest of the paper.

Theorem 3. *If* $\mathbf{P}$ *pebbles any subset of* αn *initially unpebbled sinks of* $G_{(n,k,\alpha,\beta)}$*, starting with* $|P_0| \leq \gamma n$ *and using* $S(\mathbf{P}) \leq \gamma n$ *space, then* $T(\mathbf{P}) \geq 2^k \alpha n$*.*

Proof. For the base case $k = 0$, $G_{(n,0,\alpha,\beta)}$ is simply a collection of n isolated vertices with no edges. Each vertex is both a source and a sink. The theorem is trivially true since the αn initially unpebbled sinks have to be pebbled.

Now we show the inductive step for $k \geq 1$ assuming the theorem holds for $k-1$. In $G_{(n,k,\alpha,\beta)}$, sinks are in V_k. The αn to-be-pebbled sinks in V_k are connected to at least βn vertices in V_{k-1} due to the (n, α, β) expander property. Out of these βn vertices in V_{k-1}, at least $\beta n - \gamma n = 2\alpha n$ of them are unpebbled initially in P_0 since $|P_0| \leq \gamma n$. These $2\alpha n$ vertices in V_{k-1} are unpebbled sinks of $G_{(n,k-1,\alpha,\beta)}$. Divide them into two groups of αn each in the order they are pebbled 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 $\mathbf{P}_1$ pebbles the first group ($\mathbf{P}_1$ does not pebble any vertex in the second group) and $\mathbf{P}_2$ pebbles the second group. Due to the inductive hypothesis, $T(\mathbf{P}_1) \geq 2^{k-1}\alpha n$. The starting configuration of $\mathbf{P}_2$ is the ending configuration of $\mathbf{P}_1$. At the end of $\mathbf{P}_1$, there are at most γn pebbles on the graph, and the second group of αn vertices are all unpebbled. So we can invoke the inductive hypothesis again, and have $T(\mathbf{P}_2) \geq 2^{k-1}\alpha n$. Therefore, $T(\mathbf{P}) = T(\mathbf{P}_1) + T(\mathbf{P}_2) \geq 2^k \alpha n$. □

Theorem 3 lower bounds the space complexity of any feasible pebbling strategy for stacked bipartite expanders to γn, where $\gamma = \beta - 2\alpha$. If we increase β or decrease α, γ improves but the in-degree d also increases due to Theorem 1. For each $\gamma = \beta - 2\alpha$, we find the α and β that minimize d, and plot it in Figure 3. The curves show the efficiency vs. space-hardness trade-offs our constructions can provide. For $\gamma < 0.7$, d is reasonably small. Beyond $\gamma = 0.9$, d starts to increase very fast. We recommend parameterizing our constructions around $0.7 \leq \gamma \leq 0.9$.

However, even if γ is close to 1, we still have a gap of 2 as our simple pebbling strategy for stacked bipartite expanders needs $2n$ space. To address this gap, we adopt the localization technique in Balloon hash [20].

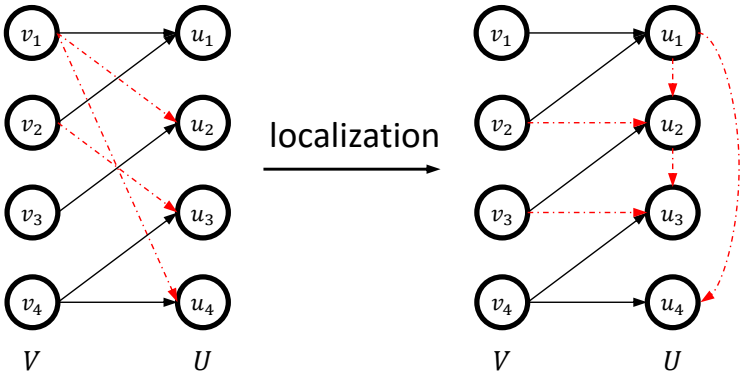

Fig. 4: Localization for a bipartite expander.

3.4 Localization of Bipartite Expanders

Localization [20] is a transformation on the edges of a bipartite expander. Consider an (n, α, β) bipartite expander with sources $V = \{v_1, v_2, \cdots v_n\}$ and sinks $U = \{u_1, u_2, \cdots u_n\}$. The localization operation first adds an edge (v_i, u_i) for all i (if it does not already exist), and then replaces each edge (v_i, u_j) where $i < j$ with (u_i, u_j). Figure 4 highlights the removed and the added edges in red. Pictorially, it adds an edge for each horizontal source-sink pair, and replaces each "downward diagonal" edge with a corresponding "downward vertical" edge. This adds at most one incoming edge for each vertex in U.

Let $LG_{(n,k,\alpha,\beta)}$ be a stack of localized expanders, i.e., the resulting graph after localizing the bipartite expander at every layer of $G_{(n,k,\alpha,\beta)}$. $LG_{(n,k,\alpha,\beta)}$ can be efficiently pebbled using n space, by simply pebbling each layer in order and within each layer from top to bottom. Once $v_{k,i}$ is pebbled, $v_{k-1,i}$ can be unpebbled because no subsequent vertices depend on it. A vertex $v_{k,j} \in V_k$ that originally depended on $v_{k-1,i}$ is either already pebbled (if $j \leq i$), or has its dependency changed to $v_{k,i}$ by the localization transformation.

When we localize a bipartite expander, the resulting graph is no longer bipartite. The expanding property, however, is preserved under a different definition. After localization, the graph has n sources and n non-sources (the original sinks). Any subset U' of αn non-sources collectively have βn sources as ancestors (v is an ancestor of u if there is a path from v to u). Crucially, the paths between them are vertex-disjoint outside U'. This allows us to prove the same result in Theorem 3 for stacked localized expanders.

Lemma 1. *Let U' be any subset of αn sinks of an (n, α, β) bipartite expander, and V' be the set of sources connected to U' (we have $|V'| \geq \beta n$). After localization, there exist βn paths from V' to U' that are vertex-disjoint outside U'.*

Proof. After localization, vertices in V' fall into two categories. A vertex $v_i \in V'$ may still be an immediate predecessor to some $u \in U'$, which obviously does

not share any vertex outside U' with a path starting from any v_j ($j \neq i$). If v_i is not an immediate predecessor, then the path $v_i \to u_i \to u$ must exist for some $u \in U'$, because there was an edge (v_i, u) in the original bipartite expander. Any other $v_j \in V'$ ($j \neq i$) is either an immediate predecessor or uses u_j as the intermediate hop in its path to U'. In either case, v_i does not share any source or intermediate-hop with any other v_j.

Theorem 4. *Let $\gamma = \beta - 2\alpha > 0$. If $\mathbf{P}$ pebbles any subset U' of αn initially unpebbled vertices in the last layer V_k of $LG_{(n,k,\alpha,\beta)}$, starting with $|P_0| \leq \gamma n$ and using $S(\mathbf{P}) \leq \gamma n$ space, then $T(\mathbf{P}) \geq 2^k \alpha n$.*

Proof. The proof remains unchanged from Theorem 3 as long as we show that $\mathbf{P}$ still needs to pebble $2\alpha n$ initially unpebbled vertices in V_{k-1}.

A path from v to u is "initially pebble-free", if no vertex on the path, including v and u, is pebbled in P_0. Due to the pebble game rule, if a vertex $v \in V_{k-1}$ has an initially pebble-free path to some $u \in U'$, then it needs to be pebbled before u can be pebbled. Since V_{k-1} and V_k form a localized expander, due to Lemma 1, there exist at least βn ancestors in V_{k-1} whose paths to U' are vertex-disjoint outside U'. Since vertices in U' are initially unpebbled, pebbles in P_0 can only be placed on the vertex-disjoint parts of these paths. Therefore, each pebble can block at most one of these paths. Since $|P_0| \leq \gamma n$, there must be at least $\beta n - \gamma n = 2\alpha n$ vertices in V_{k-1} that have initially pebble-free paths to U', and they have to be pebbled by $\mathbf{P}$. □

We now have tight space lower bounds for pebble games on stacked localized expanders. $LG_{(n,k,\alpha,\beta)}$ can be efficiently pebbled with n space but not with γn space, where γ can be set close to 1. Next, we show that pebble games on localized stacked expanders are also consistently space-hard.

3.5 Consistent Space Hardness

Theorem 5. *Let $0 < \eta < \gamma = \beta - 2\alpha$. If $\mathbf{P}$ pebbles any subset of αn initially unpebbled vertices in the last layer of $LG_{(n,k,\alpha,\beta)}$, starting with $|P_0| \leq \eta n$, and using $T(\mathbf{P}) \leq 2^{k_0} \alpha n$ moves, then*

$$M_{\eta n}(\mathbf{P}) \geq \begin{cases} 0 & k < k_0 \\ 2^{k-k_0} & k_0 \leq k \leq k_1 \\ (k - k_1 + 1)(\gamma - \eta)n & k > k_1 \end{cases}$$

where $k_1 = k_0 + \lceil \log_2(\gamma - \eta)n \rceil$.

Proof. We will derive a lower bound M_k for $M_{\eta n}(\mathbf{P})$ where k is the number of layers in $LG_{(n,k,\alpha,\beta)}$. Similar to the proof of Theorem 3, there are $(\beta - \eta)n$ initially unpebbled vertices in V_{k-1} that have to pebbled by $\mathbf{P}$. Let U be the set of these $(\beta - \eta)n$ vertices. Again, we sort U according to the time they are first pebbled. We divide $\mathbf{P}$ into three parts $\mathbf{P} = (\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$. $\mathbf{P}_1$ pebbles the first αn

vertices in $U \subset V_{k-1}$. $\mathbf{P}_1$ starts from the same initial configuration as $\mathbf{P}$ and has fewer moves than $\mathbf{P}$, so we have $M_{\eta n}(\mathbf{P}_1) \geq M_{k-1}$.

We define $\mathbf{P}_2$ to include all consecutive configurations immediately after $\mathbf{P}_1$ until (and including) the first configuration whose space usage is below ηn. $\mathbf{P}_3$ is then the rest of $\mathbf{P}$. By definition, every $P_i \in \mathbf{P}_2$, except the last one, satisfies $|P_i| > \eta n$. The last configuration in $\mathbf{P}_2$ is also the starting configuration of $\mathbf{P}_3$, and its space usage is below ηn. It is possible that $T(\mathbf{P}_2) = 1$ or $T(\mathbf{P}_3) = 0$, if the space usage after $\mathbf{P}_1$ immediately drops below ηn or never drops below ηn.

Now we have two cases based on $T(\mathbf{P}_2)$. If $T(\mathbf{P}_2) > (\gamma - \eta)n$, we have $M_k > M_{k-1} + (\gamma - \eta)n$. If $T(\mathbf{P}_2) \leq (\gamma - \eta)n$, then $\mathbf{P}_3$ has to pebble at least αn vertices in U, because $\mathbf{P}_1$ and $\mathbf{P}_2$ combined have pebbled no more than $\alpha n + (\gamma - \eta)n = (\beta - \alpha - \eta)n$ vertices in U. And $\mathbf{P}_3$ starts with no more than ηn pebbles and has fewer moves than $\mathbf{P}$, so $M_{\eta n}(\mathbf{P}_3) \geq M_{k-1}$. In this case, we have $M_k \geq 2M_{k-1}$. Combining the two cases, we have the following recurrence

$$M_k \geq \min(M_{k-1} + (\gamma - \eta)n, 2M_{k-1}).$$

For a base case of this recurrence, we have $M_{k_0} \geq 1$, because Theorem 4 says any pebbling strategy that never uses ηn space needs at least $2^{k_0}\alpha n$ moves. Solving the recurrence gives the result in the theorem. □

For a tight bound on the entire sequence, we further chain the vertices in $LG_{(n,k,\alpha,\beta)}$, by adding an edge $(v_{i,j}, v_{i,j+1})$ for every $0 \leq i \leq k$ and $1 \leq j \leq n-1$. (We can prove a looser bound without the chaining technique.) This forces any sequence to pebble all vertices in the same order as the simple strategy.

Corollary 1. *Any sequence* $\mathbf{P}$ *that pebbles the chained stacked localized expanders* $LG_{(n,k,\alpha,\beta)}$ *starting from an empty initial configuration in* $T(\mathbf{P}) \leq 2^{k_0}\alpha n$ *steps has* $M_{(\beta-3\alpha)n}(\mathbf{P}) \geq n(k - k_1)$ *where* $k_1 = k_0 + \lceil \log_2(\alpha n) \rceil$.

Proof. Set $\eta = \beta - 3\alpha$. Theorem 5 shows that beyond the first k_1 layers, it is expensive to ever reduce space usage below ηn. Doing so on layer $k > k_1$ would require at least $(k - k_1 + 1)\alpha n > \alpha n$ steps with ηn space usage to pebble the next αn vertices. The penalty keeps growing with the layer depth. So the better strategy is to maintain space usage higher than ηn for every step past layer k_1. There are at least $n(k - k_1)$ steps past layer k_1, and hence the theorem holds. □

The simple strategy maintains n space for nk steps, i.e., the entire duration except for the first n steps which fill memory. Corollary 1 is thus quite tight as $n(k - k_1)$ and ηn can be very close to nk and n with proper parameters.

4 Improved Analysis for Balloon Hash MHF

A memory-hard function (MHF) is a function f that (i) takes a short input, and (ii) requires a specified amount of, say N, space to compute efficiently. To our knowledge, all MHF proposals first put the input through a hash function $\mathcal{H}$ so that $f(\mathcal{H}(\cdot))$ can take input of any size, and $f(\cdot)$ only deals with a fixed input

Table 1: Comparison of MHFs with strict space-hardness or exponential penalty.

	DKW [26]	KK [31]	Catena-DBG [28]	Balloon [20]	Balloon + our analysis
T	N^2	$2N(\log_2 N)^2$	$2kN \log_2 N$	$7kN$	dkN
N'	N	$N/32$	$N/20$	$N/8$	γN

KK [31] reported $T = \Theta(N \log_2 N)$ due to a miscalculation on the number of vertices. We generalized Catena-DBG [28] to exponential penalty though the designers of Catena recommended tiny security parameters like $k = 2$ for better efficiency.

size $\lambda = |\mathcal{H}(\cdot)|$. λ is considered short since it does not depend on N. There is no agreed upon criterion of memory-hardness. As discussed in Section 1.1, we adopt the exponential penalty definition.

Definition 2 (MHF). *Let k be a security parameter, N be the space requirement, and N' be the provable space lower bound. A memory-hard function $y = f(x)$, parameterized k, N and N', has the following properties:*

(non-triviality) *the input size $|x|$ does not depend on N,*
(efficiency) *f can be computed using N space in $T = \mathsf{poly}(k, N)$ time,*
(memory-hardness) *no algorithm can compute f using less than N' space in 2^k time with non-negligible probability.*

The graph labelling problem on a hard-to-pebble graph immediately gives a MHF. Table 1 lists the running time T and the provable space lower bound N' for all existing MHFs with strict memory-hardness or exponential penalty (though the former two did not use the term MHF). All of them are based on graph pebbling. DKW [26] has perfect memory-hardness but requires a quadratic runtime. The other three have quasilinear runtime but large gaps in memory-hardness. The single-buffer version of Balloon hash (Balloon-SB) [20] used stacked localized expanders. Using the analysis in the Balloon hash paper, the space lower bound N' for Balloon-SB is at most $N/4$ no matter how much we sacrifice runtime. Our improved analysis shows that Balloon-SB enjoys tighter space-hardness as well as consistent space-hardness. Theorem 4 shows that Balloon-SB with $T = dkN$ achieves $N' = \gamma N$, where γ can be made arbitrarily small. The relation between γ and d is shown in Figure 3. In addition, Corollary 1 gives a tight bound on consistent memory-hardness. This gives positive answers to two open questions left in the Balloon hash paper [20]: Balloon hash is space-hard over time and under batching.

5 Proofs of Transient Space from Stacked Expanders

5.1 Definition

We use notation $(y_v, y_p) \leftarrow \langle \mathsf{V}(x_v), \mathsf{P}(x_p) \rangle$ to denote an interactive protocol between a verifier V and a prover P. x_v, x_p, y_v, y_p are V's input, P's input,

V's output and P's output, respectively. We will omit (x_v) or (x_p) if a party does not take input. We will omit y_p if P does not have output. For example, $\{0,1\} \leftarrow \langle \mathsf{V}, \mathsf{P} \rangle$ means neither V nor P takes input, and V outputs one bit indicating if it accepts (output 1) or rejects (output 0) P's proof. Both P and V can flip coins and have access to the same random oracle $\mathcal{H}$.

Definition 3 (PoTS). *Let k, N and N' be the same as in Definition 2. A proof of transient space is an interactive protocol $\{0,1\} \leftarrow \langle \mathsf{V}, \mathsf{P} \rangle$ that has the following properties:*

(succinctness) *all messages between P and V have size $\mathsf{poly}(k, \log N)$,*
(efficient verifiability) *V uses $\mathsf{poly}(k, \log N)$ space and time,*
(completeness) *P uses N space, runs in $\mathsf{poly}(k, N)$ time, and $\langle \mathsf{V}, \mathsf{P} \rangle = 1$,*
(space-hardness) *there does not exist A that uses less than N' space, runs in 2^k time, and makes $\langle \mathsf{V}, \mathsf{A} \rangle = 1$ with non-negligible probability.*

The definition above is due to Ateniese et al. [11]. Metrics for a PoTS include message size, prover runtime, verifier space/runtime, and the gap between N and N'. The first three measure efficiency and the last one measures space-hardness. We also care about consistent space-hardness as defined in Section 3.5.

5.2 Construction

We adopt the Merkle commitment framework in Ateniese et al. [11] and Dziembowski et al. [24] to enable efficient verification. At a high level, the prover computes a Merkle commitment C that commits the labels of all vertices in $LG_{(n,k,\alpha,\beta)}$ using the same random oracle $\mathcal{H}$. The verifier then checks if C is "mostly correct" by asking the prover to open the labels of some vertices. The opening of label $h(v)$ is the path from the root to the leaf corresponding to v in the Merkle tree. To compute a commitment C that is "mostly correct", a prover cannot do much better than pebbling the graph following the rules, which we have shown to require a lot of space consistently. We say "a vertex" instead of "the label of a vertex" for short. For example, "commit/open a vertex" means "commit/open the label of a vertex".

Computing a Merkle tree can be modeled as a pebble game on a binary tree graph. It is not hard to see that a complete binary tree with n leaves can be efficiently pebbled with roughly $\log_2 n$ space ($\lceil \log_2 n \rceil + 1$ to be precise) in n moves. So P can compute the commitment C using $N = n + \log_2 n + k \approx n$ space. The strategy is as follows: pebble V_0 using n space, compute Merkle commitment C_0 for all vertices in V_0 using additional $\log_2 n$ space, discard the Merkle tree except the root, and then pebble V_1 rewriting V_0, compute C_1, discard the rest of the Merkle tree, and continue like this. Lastly, C_1 to C_k are committed into a single Merkle root C.

After receiving C, V randomly selects l_0 vertices, and for each vertex v asks P to open v, and all predecessors of v if v is not a source. Note that P did not store the entire Merkle tree but was constantly rewriting parts of it because the entire tree has size $nk \gg n$. So P has to pebble the graph for a second time

Table 2: Efficiency and space-hardness of PoTS.

	prover runtime	verifier space/time and message size	N'
ABFG [11]	$12kN \log_2 N$	$6\delta^{-1}k^2(\log_2 N)^2$	$(\frac{1}{6} - \delta)N$
This paper	$2(d+1)kN$	$(d+1)\delta^{-1}k^2 \log_2 N$	$(\gamma - \delta)N$

to reconstruct the $l_0(d+1)$ paths/openings V asked for. This is a factor of 2 overhead in prover's runtime.

Given the labels of all the predecessors of v (or if v is a source), V can check if $h(v)$ is correctly computed. If any opening or $h(v)$ is incorrect, V rejects. If no error is found, then C is "mostly correct". We say a label $h(v_i)$ is a "fault" under C if it is not correctly computed either as $h(i, x)$ or from v_i's predecessors' labels under C. A cheating prover is motivated to create faults using pseudorandom values, because these faulty labels are essentially free pebbles that are always available but take no space. Dziembowski et al. [24] called them red pebbles and pointed out that a red pebble is no more useful than a free normal pebble because a normal pebble can be removed and later placed somewhere else. In other words, any sequence $\mathbf{P}$ that starts with $|P_0| = s_0$ initial pebbles and uses m red pebbles and s normal pebbles can be achieved by some sequence $\mathbf{P'}$ that starts with $|P'_0| = s_0 + m$ initial pebbles and uses 0 red pebbles and $s + m$ normal pebbles. We would like to bound the number of faults, which translate to a bounded loss in provable space-hardness.

If we want to lower bound the number of faults to δn ($\delta < 1$) with overwhelming probability, we can set $l_0 = \frac{k|V|}{\delta n} = \delta^{-1}k^2$. Then, any commitment C with δn faults passes the probabilistic checking with at most $(1 - \frac{\delta n}{|V|})^{l_0} < e^{-k}$. Again, k is our security parameter. With at most δn faults, P needs to pebble at least $n - \delta n$ sinks ($> \alpha n$ with a proper δ). By Theorem 4 and accounting for faults, a cheating prover needs at least $N' = (\gamma - \delta)n \approx (\gamma - \delta)N$ space to avoid exponential time.

5.3 Efficiency and Space-Hardness

Table 2 gives the efficiency and space-hardness of our construction, and compares with prior work using stacked butterfly superconcentrators [11]. Our prover runtime is $2(d+1)Nk$ where 2 is due to pebbling the graph twice, and $d+1$ is due to the in-degree of our graph plus hashing in Merkle tree. Message size includes Merkle openings for the $l_0 = \delta^{-1}k^2$ challenges and their predecessors. The verifier has to check all these Merkle openings, so its space/time complexity are the same as message size. The efficiency of ABFG [11] can be calculated similarly using the fact that stacked butterfly superconcentrators have $2kN \log N$ vertices with in-degree 2. To match their space-hardness, which cannot be improved past $N' = \frac{1}{6}N$ with existing proof techniques, we only need in-degree $d = 9$. To match their efficiency, we set $d = 6 \log_2 N$, which we approximate as 150. That gives our construction very tight space-hardness at $N' = (\gamma - \delta)N$ with $\gamma = 0.85$.

Furthermore, Corollary 1 gives a tight bound on consistent memory-hardness. Adjusting for faults, an adversary needs $n(k - k_1)$ steps whose space usage is at least $(\beta - 3\alpha - \delta)n$.

For simplicity, we used a single security parameter k. But in fact, the term k^2 in message size and verifier complexity should be kk' where k' is a statistical security parameter. k' can be set independently from our graph depth k, which captures computational security. The same applies to the DFKP construction in Table 3.

6 Proof of Persistent Space from Stacked Expanders

6.1 Definition

Definition 4 (PoPS).
Let k be a security parameter, N be the space/advice requirement, N'_0 and N'_1 be two space lower bound parameters. A proof of persistent space is a pair of interactive protocols $(C, y) \leftarrow \langle \mathsf{V}_0, \mathsf{P}_0 \rangle$ and $\{0,1\} \leftarrow \langle \mathsf{V}_1(C), \mathsf{P}_1(y) \rangle$ that have the following properties:

(succinctness) *all messages between P_0 and V_0, and between P_1 and V_1 have size $\mathsf{poly}(k, \log N)$,*
(efficient verifiability) *V_0 and V_1 use $\mathsf{poly}(k, \log N)$ space and time,*
(completeness) *P_0 and P_1 satisfy the following*
- *P_0 uses N space, runs in $\mathsf{poly}(k, N)$ time, and outputs y of size N,*
- *P_1 uses N space, runs in $\mathsf{poly}(k, \log N)$ time, and $\langle \mathsf{V}_1(C), \mathsf{P}_1(y') \rangle = 1$,*

(space-hardness) *there do not exist A_0 and A_1 such that*
- *A_0 uses $\mathsf{poly}(k, N)$ space, runs in $\mathsf{poly}(k, N)$ time, and $\langle \mathsf{V}_0, \mathsf{A}_0 \rangle = (C, y')$ where $|y'| < N'_0$,*
- *A_1 takes y' as input, uses N'_1 space, runs in 2^k time, and makes $\langle \mathsf{V}_1(C), \mathsf{A}_1(y') \rangle = 1$ with non-negligible probability.*

$(C, y) \leftarrow \langle \mathsf{V}_0, \mathsf{P}_0 \rangle$ represents the setup stage. P outputs advice y of size N, which is supposed to be stored persistently. V (through interaction with P) outputs a verified commitment C. $\{0,1\} \leftarrow \langle \mathsf{V}_1(C), \mathsf{P}_1(y) \rangle$ represents one audit. The input of two parties are their respective output from the setup stage, and in the end V either accepts or rejects. It is implied that an audit V_1 has to use random challenges. Otherwise, it is easy to find A_1 that takes as input and also outputs the correct response to a fixed audit.

Efficiency metrics (message size, prover runtime, verifier space/runtime) are defined similarly to PoTS but now take into account both stages of the protocol.

The space-hardness definition and metric become a little tricky. Since the focus here is persistent space or advice size, one natural definition is to require that no polynomial adversary A_1 with advice size $|y'| < N'_0$ can convince V with non-negligible probability. Unfortunately, this ideal space-hardness definition is not achievable given the succinctness requirement, and we will describe an adversary who can violate it. In the setup phase, A_0 behaves in the same way as

an honest P_0 except that it outputs the transcript (all the messages combined) between A_0 and V_0 as the cheating advice y'. Due to succinctness, the transcript size $|y'| = \mathsf{poly}(k, \log N)$ is much shorter than any reasonable N'_0. In an audit, A_1 can rerun P_0 by simulating V_0, using the recorded transcript y', to obtain the advice y that P_0 would have produced, and then go on to run an honest P_1 to pass the audit.

Given the impossibility of ideal space-hardness, multiple alternative definitions have been proposed. Dziembowski et al. [24] gave two definitions. The first one requires A_1 to use $O(N)$ space. The second one requires A_1 to use $O(N)$ runtime, which is strictly weaker than the first one because $O(N)$ transient space implies $O(N)$ runtime. Proof of Space-Time [35] introduces a conversion rate between space and computation, and requires the sum of the two resources spent by A_1 to be within a constant factor of the sum spent by P_0. In this paper, we adopt the first definition of Dziembowski et al. [24] because it forces a prover to use space (either transient or persistent). In contrast, the latter two definitions explicitly allow a cheating prover to use tiny space and compensate with computation.

Under our space-hardness definition, if a cheating prover discards persistent advice in an attempt to save space, he/she will find himself/herself repeatedly refilling that space he/she attempts to save. If (N'_0, N'_1) are close to (N, N), a rational prover will choose to dedicate persistent space for the advice. We would like to be explicit that such a PoPS relies on a prover's cost of persistent space relative to computation and transient space, and very importantly the frequency of audits.

6.2 Construction

The setup phase is basically the PoTS protocol we presented in Section 5. P computes a Merkle commitment C, and V makes sure C is "mostly correct" through a probabilistic check. At the end of the setup phase, an honest P stores the labels of all sinks V_k and the Merkle subtree for V_k as advice. Any vertices in V_i for $i < k$ are no longer needed. V now can also discard C and use C_k which commits V_k from this point onward. Since an honest P has to store the Merkle tree, it makes sense to use a different random oracle $\mathcal{H}_1$ with smaller output size for the Merkle commitment. If $|\mathcal{H}(\cdot)|$ is reasonably larger than $|\mathcal{H}_1(\cdot)|$, then the labels in the graph dominate, and the advice size is thus roughly n. Using the same random oracle results in an additional factor of 2 loss in space-hardness.

In the audit phase, V asks P to open l_1 randomly selected sinks. The binding property of the Merkle tree forces P to pebble these sinks, possibly with the help of at most δn faults. But due to the red pebble argument, we can focus on the case with no faults first and account for faults later.

There is still one last step from Theorem 4 to what we need. Theorem 4 says any subset of αn initially unpebbled sinks are hard to pebble, but we would hope to challenge P on $l_1 \ll \alpha n$ sinks. Therefore, we need to show that a significant fraction of sinks are also hard to pebble individually.

Table 3: Efficiency and space-hardness of PoPS.

		prover runtime	verifier space/runtime and message size	N'_0 N'_1
DFKP [24]	setup	$3N$	$3\delta^{-1}k(\log_2 N)^2$	$(\frac{1}{3} \times \frac{1}{256\times 25.3} - \delta)\frac{N}{\log_2 N}$
	audit	0	$k \log_2 N$	$(\frac{2}{3} \times \frac{1}{256\times 25.3} - \delta)\frac{N}{\log_2 N}$
This paper	setup	$2(d+1)kN$	$(d+1)\delta^{-1}k^2 \log_2 N$	$(\frac{1}{3}\gamma - \delta)N$
	audit	0	$k \log_2 N$	$(\frac{2}{3}\gamma - \delta)N$

Theorem 6. *Let $\gamma = \beta - 2\alpha$. Starting from any initial configuration P_0 of size $|P_0| \leq \frac{1}{3}\gamma n$, less than αn initially unpebbled sinks of $G_{(n,k,\alpha,\beta)}$ can be pebbled individually using $\frac{2}{3}\gamma n$ space in 2^k moves.*

Proof. Suppose for contradiction that there are at least αn such sinks. Consider a strategy that pebbles these sinks one by one, never unpebbles P_0, and restarts from P_0 after pebbling each sink. This strategy pebbles a subset of αn initially unpebbled sinks, starting with $|P_0| < \frac{1}{3}\gamma n < \gamma n$, using at most $\frac{1}{3}\gamma n + \frac{2}{3}\gamma n = \gamma n$ pebbles in at most $2^k \alpha n$ moves. This contradicts Theorem 4. □

At most $\frac{1}{3}\gamma n$ pebbles may be initially pebbled in P_0, so no more than $(\frac{1}{3}\gamma + \alpha)n < \frac{1}{2}n$ individual sinks can be pebbled using $\frac{2}{3}\gamma n$ space in 2^k moves by Theorem 6. With more than half of the sinks being hard to pebble individually, we can set $l_1 = k$. The probability that no hard-to-pebble sink is included in the challenge is at most 2^{-k}. Lastly, accounting for faults, no computationally bounded P using $N'_0 = (\frac{1}{3}\gamma - \delta)n$ advice and $N'_1 = (\frac{2}{3}\gamma - \delta)n$ space can pass an audit. The choice of constants $\frac{1}{3}$ and $\frac{2}{3}$ are arbitrary. The theorem holds for any pair of constants that sum to 1.

6.3 Efficiency and Space-Hardness

We compare with prior work [24] based on recursively stacked linear superconcentrators [38] in Table 3. The efficiency and (consistent) space-hardness of the setup phase are the same as our PoTS. In the audit phase, the prover sends Merkle openings for k sinks to the verifier to check. If the prover stores less than $N'_0 = (\frac{1}{3}\gamma - \delta)N$ advice, it needs at least $N'_1 = (\frac{2}{3}\gamma - \delta)N$ space to pass an audit. This also sets a lower bound of $N'_1 - N'_0 = (\frac{1}{3}\gamma - \delta)N$ on prover's time to pass the audit, as it needs to fill its space to N'_1. Consistent space-hardness is not well defined for audits as an honest prover needs very little time to respond to audits.

The DFKP construction [24] treats the labels of all vertices as advice. This optimizes runtime but leaves a very large (even asymptotic) gap in space-hardness. It is possible for them to run audits only on the sinks, essentially generating less advice using the same graph and runtime. This will improve space-hardness up to $N'_0 = N'_1 = (\frac{1}{2} \times \frac{1}{256} - \delta)N$ while increasing runtime by a factor of $25.3 \log N$. There is currently no known way to remove the remaining gap of 512. We did not count the cost of including vertex ID in random oracle calls (for

both our construction and theirs) since vertex ID is small compared to labels. This explains the different prover runtime we report in Table 3 compared to [24]. For completeness, we mention that the second construction of DFKP [24] and proof of Space-Time [35] are not directly comparable to our construction or the first DFKP construction because they provide very different efficiency and/or space-hardness guarantees.

7 Conclusion and Future Work

We derived tight space lower bounds for pebble games on stacked expanders, and showed that a lot of space is needed not just at some point, but throughout the pebbling sequence. These results gave MHF (Balloon hash) and PoTS with tight and consistent space-hardness. We also constructed a PoPS from stacked expanders with much better space-hardness than prior work.

While the space-hardness gap for Balloon hash and our PoTS can be made arbitrarily small, pushing it towards the limit would lead to very large constants for efficiency. How to further improve space-hardness for MHF and PoS remains interesting future work. It is also interesting to look for constructions that maintain consistent space-hardness under massive or even infinite parallelism.

At the moment, PoTS and PoPS are still far less efficient than PoW in terms of proof size and verifier complexity. A PoW is a single hash, while a PoS consists of hundreds (or more) of Merkle paths. The challenge remains in constructing practical PoTS/PoPS with tight and consistent space-hardness.

References

1. Zoom Hash Scrypt ASIC. `http://zoomhash.com/collections/asics`. Accessed: 2016-05-20.
2. Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions. *ACM Transactions on Internet Technology*, 5(2):299–327, 2005.
3. Leonardo C. Almeida, Ewerton R. Andrade, Paulo S. L. M. Barreto, and Marcos A. Simplicio Jr. Lyra: Password-based key derivation with tunable memory and processing costs. *Journal of Cryptographic Engineering*, 4(2):75–89, 2014.
4. Noga Alon and Michael Capalbo. Smaller explicit superconcentrators. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 340–346. Society for Industrial and Applied Mathematics, 2003.
5. Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. Cryptology ePrint Archive, Report 2016/115, 2016.
6. Joël Alwen and Jeremiah Blocki. Towards practical attacks on argon2i and balloon hashing. Cryptology ePrint Archive, Report 2016/759, 2016.
7. Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. On the complexity of scrypt and proofs of space in the parallel random oracle model. In *Advances in Cryptology–EUROCRYPT 2016*, pages 358–387. Springer, 2016.

8. Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 595–603. ACM, 2015.
9. David G. Andersen. Exploiting time-memory tradeoffs in cuckoo cycle, 2014. (Accessed August 2016) https://www.cs.cmu.edu/ dga/crypto/cuckoo/analysis.pdf.
10. Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, 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.
11. Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: when space is of the essence. In *Security and Cryptography for Networks*, pages 538–557. Springer, 2014.
12. Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609. ACM, 2007.
13. Adam Back. Hashcash-a denial of service counter-measure, 2002.
14. Leonid Alexandrovich Bassalygo. Asymptotically optimal switching circuits. *Problemy Peredachi Informatsii*, 17(3):81–88, 1981.
15. Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Fast and tradeoff-resilient memory-hard functions for cryptocurrencies and password hashing, 2015.
16. Alex Biryukov and Dmitry Khovratovich. Tradeoff cryptanalysis of memory-hard functions. Cryptology ePrint Archive, Report 2015/227, 2015.
17. Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. In *NDSS*, 2016.
18. F.R.K. Chung. On concentrators, superconcentrators, generalizers, and nonblocking networks. *Bell System Technical Journal*, 58(8):1765–1777, 1979.
19. Stephen A. Cook. An observation on time-storage trade off. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 29–33. ACM, 1973.
20. Henry Corrigan-Gibbs, Dan Boneh, and Stuart Schechter. Balloon hashing: a provably memory-hard function with a data-independent access pattern. Cryptology ePrint Archive, Report 2016/027, 2016.
21. Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *Advances in Cryptology–Crypto 2003*, pages 426–444. Springer, 2003.
22. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology–CRYPTO 1992*, pages 139–147. Springer, 1992.
23. Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In *Advances in Cryptology–CRYPTO 2005*, pages 37–54. Springer, 2005.
24. Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Advances in Cryptology–CRYPTO 2015*, pages 585–605. Springer, 2015.
25. Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. Key-evolution schemes resilient to space-bounded leakage. In *Advances in Cryptology–CRYPTO 2011*, pages 335–353. Springer, 2011.
26. Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In *Theory of Cryptography*, pages 125–143. Springer, 2011.
27. Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Overview of the candidates for the password hashing competition, 2015.

28. Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena : A memory-consuming password-scrambling framework. Cryptology ePrint Archive, Report 2013/525, 2013.
29. John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space and related problems. In *16th Annual Symposium on Foundations of Computer Science*, pages 57–64. IEEE, 1975.
30. Ari Juels and Burton S. Kaliski Jr. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597. ACM, 2007.
31. Nikolaos P. Karvelas and Aggelos Kiayias. Efficient proofs of secure erasure. In *Security and Cryptography for Networks*, pages 520–537. Springer, 2014.
32. Thomas Lengauer and Robert E. Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *Journal of the ACM*, 29(4):1087–1130, 1982.
33. Sergio Demian Lerner. Strict memory hard hashing functions (preliminary v0. 3, 01-19-14).
34. Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 373–388. ACM, 2013.
35. Tal Moran and Ilan Orlov. Proofs of space-time and rational proofs of storage. Cryptology ePrint Archive, Report 2016/035, 2016.
36. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
37. Wolfgang J. Paul and Robert E. Tarjan. Time-space trade-offs in a pebble game. *Acta Informatica*, 10(2):111–115, 1978.
38. Wolfgang J. Paul, Robert E. Tarjan, and James R. Celoni. Space bounds for a game on graphs. *Mathematical Systems Theory*, 10(1):239–251, 1976.
39. Colin Percival. Stronger key derivation via sequential memory-hard functions, 2009.
40. Daniele Perito and Gene Tsudik. Secure code update for embedded devices via proofs of secure erasure. In *ESORICS*, pages 643–662. Springer, 2010.
41. Alexander Peslyak. yescrypt - a password hashing competition submission, 2014. (Accessed August 2016) https://password-hashing.net/submissions/specs/yescrypt-v2.pdf.
42. Mark S Pinsker. On the complexity of a concentrator. In *7th International Telegraffic Conference*, volume 4, 1973.
43. Herbert Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62(1):26–29, 1955.
44. Uwe Schöning. Better expanders and superconcentrators by kolmogorov complexity. In *SIROCCO*, pages 138–150, 1997.
45. Uwe Schöning. Smaller superconcentrators of density 28. *Information processing letters*, 98(4):127–129, 2006.
46. Ravi Sethi. Complete register allocation problems. *SIAM journal on Computing*, 4(3):226–248, 1975.
47. Adam Smith and Ye Zhang. Near-linear time, leakage-resilient key evolution schemes from expander graphs. Cryptology ePrint Archive, Report 2013/864, 2013.
48. John Tromp. Cuckoo cycle: a memory-hard proof-of-work system, 2014.