

Proof of Space from Stacked Bipartite Graphs

Ling Ren

Srinivas Devadas

Abstract

This report presents a proof of space protocol based on pebble games on stacked bipartite graphs.

1 introduction

A proof of space is a protocol between a *prover* and a *verifier*, that convinces the verifier that the prover has dedicated some amount of (memory or disk) space. Most recent proposals are based on hard-to-pebble graphs, especially stacked *superconcentrators*. These graphs either add a logarithmic factor overhead to the protocol, or are hard to construct and have bad constants. In this report, we construct proof of space protocols from a simpler family of hard-to-pebble graphs, namely *stacked bipartite graphs*.

Somewhat unfortunately, two competing definitions of “proof of space” have been proposed with very different security guarantees and applications [1, 4]. Adding to the confusion are other closely related notions such as proof of secure erasure (PoSE) [10], provable data possession (PDP) [2] and proof of retrievability (PoR) [5]. A key difference lies in whether the proof is for transient space or persistent space. We distinguish the two notions of space and define two protocols: *proof of transient space (PoTS)* and *proof of persistent space (PoPS)*.

Proof of space by Ateniese et al. [1] and proofs of secure erasure [10, 6] fall into PoTS, while proofs of space by Dziembowski et al. [4] are PoPS. PDP and PoR, when used correctly, can be both PoTS and PoPS, but require a long initial message from the verifier to the prover. PoTS’s main application is remote device wipe, while efficiently verifiable PoPS is a potential alternative to proof of work [3].

2 Definitions

Let P be the prover and V be the verifier, and N be the size of transient or persistent space P wants to prove. We assume both P and V are computationally bounded.

2.1 Proof of Transient Space

A PoTS protocol has the following steps: (1) V sends input x to P ; (2) P sends V a proof ϕ , which takes N space to generate; (3) V verifies the validity of ϕ .

The main efficiency metrics of a PoTS are message size, prover’s runtime and verifier complexity. Message size refers to the size of x and ϕ . Prover’s runtime is the time it takes an honest P to generate the proof (the space needed should of course be N). Verifier complexity is the time and space it takes V to verify the proof.

A PoTS protocol should be sound: if P uses less than N' space, V accepts the proof with only negligible probability. Ideally, we would like $N' = N$, but known schemes with this strong soundness guarantee have large message size or verifier complexity. In some scenarios, one may wish to trade off soundness for better efficiency, and tolerate a space gap of $N/N' > 1$.

Example: A trivial PoTS would simply let $x \leftarrow \{0, 1\}^n$ and $\phi = x$, i.e., V sends a long random (incompressible) string to P , asks P to send it back and then checks equality. Such a scheme has no space gap, but its message size is $2N$ and the verifier's space and time complexity are both N .

2.2 Proof of Persistent Space

We call persistent space *advice*. Similar to proof of space formulated by Dziembowski et al. [4], a PoPS protocol in this report has two phases: setup and audit. The audit phase can be executed an arbitrary number of times after setup. A PoPS ideally proves that P keeps N advice across audits.

Setup:

1. V sends input x to P .
2. Using x , P computes *advice* y of size N and a commitment C . P keeps y , and sends C to V .
3. V checks the validity of C , possibly through interaction with P .

Audit:

1. V sends a challenge c to P .
2. Using the advice y it stores from the setup phase, P computes and sends V a response r .
3. V verifies r against the commitment C it received earlier, and either accepts or rejects.

Efficiency metrics are defined similarly to PoTS. Message size now includes all messages between P and V . Verifier complexity and prover's runtime also take into account all steps of the protocol.

Soundness is a little tricky. One way to define soundness is to require that a prover P storing advice y' of size $|y'| < N'$ (again we are allowing a gap for the amount of advice) cannot generate a valid response r that passes an audit with non-negligible probability. However, such a soundness requirement immediately imposes a dilemma between message size and advice gap. To get an advice gap of N/N' , $|x|$ must be at least N' . If $|x| < N'$, a cheating prover can simply store x and rerun the setup phase every time it gets audited. (We remark that a PDP or PoR protocol on a long and incompressible x of size $|x| = N'$ does meet the above soundness requirement.)

Indeed, if we desire a PoPS with a short x , the best soundness guarantee we can hope to offer is to force a cheating prover (that stores less than N' advice) to rerun the setup phase. Following Dziembowski et al. [4], we say a PoPS is (N', S', T') -sound if no cheating prover using N' advice, S' space, and T' time can pass an audit with non-negligible probability. If the setup phase takes an honest prover S space and T time, we would like (N', S', T') to be close to (N, S, T) . If repeatedly spending S' space and T' time on audits is expensive, a rational prover will choose to dedicate persistent storage for y across audits, making the protocol a PoPS. Again, we would like to be explicit that such a PoPS relies on a prover's cost of storage relative to computation, and very importantly the frequency of audits.

3 Background on Graph Pebbling

Graph pebbling is a single-player game on a directed acyclic graph (DAG) $G = (V, E)$ with 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 (i.e., removing pebbles is free in our model). The player’s moves can be represented as a sequence of pebble placement configurations on the graph, $\mathbf{P} = (P_0, P_1, P_2 \cdots, P_T)$, starting from an initial configuration P_0 which is not necessarily empty. 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) a vertex $v \in V$ if v is a source or if all predecessors of v have pebbles on them in P_i , and then *unpebble* (i.e., remove pebbles from) any subset of vertices. Let $|P_i|$ be the number of pebbles on the graph in configuration P_i . We define the space complexity $S(\mathbf{P})$ to be $\max_i(|P_i|)$, the maximum number of pebbles on the graph in any step, and the time complexity $T(\mathbf{P})$ to be the number of moves in \mathbf{P} .

We assume we can force a prover to follow the pebble game rule using a random oracle \mathcal{H} . Following prior work, we number the vertices, and associate each vertex v_i with a label $h(v_i) \in \{0, 1\}^k$ where k is a security parameter. If v_i is not a source, define $h(v_i) = \mathcal{H}(h(u_1), h(u_2), \dots, h(u_d))$ where u_1 to u_d are the predecessors of v_i ; otherwise, define $h(v_i) = \mathcal{H}(x, i)$ where x is a nonce (which will be the input x in our PoTS and PoPS protocols).

Pebble games are good candidates for constructing proofs of space because pebble games on some families of graphs have high space complexity or sharp space-time trade-offs. These include stacked superconcentrators [9, 7], which were adopted in prior PoSE and proof of space schemes [6, 1, 4]. We notice that a simpler family of graphs—stacked random bipartite graphs—also have sharp space-time trade-offs [8] and can be used to construct proof of space protocols.

Definition 1. *An (n, β, α) bipartite expander is a directed bipartite graph with n sources and n sinks such that any subset of βn sinks are connected to more than αn sources.*

Lemma 1. *A random directed bipartite graph with n sources, n sinks and in-degree 16 is an $(n, \frac{1}{8}, \frac{1}{2})$ bipartite expander with overwhelming probability.*

Proof. Paul and Tarjan [8] showed that the probability is at least $1 - 2^{-2n} \binom{n}{n/2} \binom{n}{n/8}$. Using Robbins’ inequality for Sterling’s approximation [11], the probability is at least $1 - 2^{-0.45n/n}$. \square

Let E_n be an $(n, \frac{1}{8}, \frac{1}{2})$ bipartite expander above. Construct $G_{n,k}$ by stacking E_n . $G_{n,k}$ has nk vertices, partitioned into k sets each of size n , $V = \{V_1, V_2, \dots, V_k\}$. All edges in $G_{n,k}$ go from V_{i-1} to V_i for some i from 2 to k . For every i from 2 to k , V_{i-1} and V_i plus all edges between them form a copy of the same E_n . Thus, $G_{n,k}$ has n sources and n sinks, and in-degree at most 16. Figure 1 is an example with $n = 4$, $k = 4$ and in-degree 2.

Obviously, simply pebbling each expander in order results in a sequence \mathbf{P} that pebbles all sinks of $G_{n,k}$ using $S(\mathbf{P}) = 2n$ pebbles in $T(\mathbf{P}) = nk$ moves. Paul and Tarjan showed the following sharp space-time trade-off that will allow us to construct PoPS and PoTS in the next section.

Lemma 2 (Paul and Tarjan [8]). *Assume 8 divides n . If \mathbf{P} pebbles any subset of $n/4$ sinks of $G_{n,k}$, starting with $|P_0| < n/8$ and using no more than $S(\mathbf{P}) < n/8$ pebbles, then $T(\mathbf{P}) > 2^{k-1}n/8$.*

We have slightly tightened the base case for the inductive proof in [8]: for $k = 1$, obviously $n/4 - |P_0| > n/8$ moves are needed.

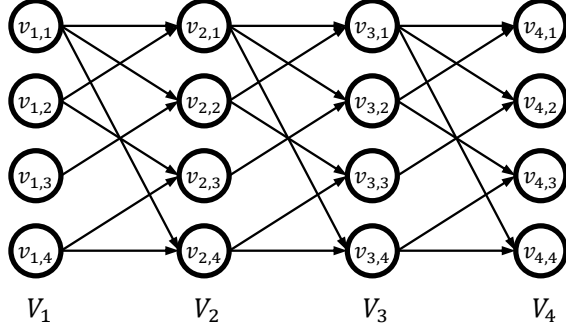


Figure 1: A stacked bipartite graph $G_{4,4}$. Each bipartite graph is a $(4, \frac{1}{4}, \frac{1}{2})$ expander.

4 PoTS and PoPS from Stacked Bipartite Graphs

As a warm-up, we directly apply Lemma 2 to obtain a simple PoTS that has small message size but large verifier complexity. V sends P a short input x as a nonce for labeling source vertices, i.e., $h(v_i) = \mathcal{H}(x, i)$. The proof ϕ can be a hash on all sinks' labels. V can then pebble $G_{n,k}$ itself and check ϕ . In this protocol, an honest P needs $N = 2n$ space (measured by label size); a cheating prover using less than $N' = n/8$ space will run for exponential time. The space gap of soundness is thus $N/N' = 16$. Efficiency wise, the messages x and ϕ are both short, the prover's runtime is nk and the verifier's complexity is the same as the prover's.

Now, to achieve small verifier complexity, we adopt the Merkle commitment framework in Ateniese et al. [1] and Dziembowski et al. [4]. We only present the PoPS protocol, because our PoTS is simply the setup phase of our PoPS. We first give an overview. In the setup phase of PoPS, the prover computes a Merkle commitment C for the labels of all vertices in $G_{n,k}$ using the same random oracle \mathcal{H} , and sends it to the verifier. The verifier checks if C is “mostly correct”. In the audit phase, the verifier asks the prover to open the labels of a few sinks under the commitment C . If C is “mostly correct”, the prover cannot do much better than pebbling these sinks following the pebble game rule. We will show that a large fraction of the sinks require $\Theta(n)$ pebbles to pebble efficiently, forcing the prover to dedicate $\Theta(n)$ space (measured by label size).

When the context is clear, we will simply say “a vertex” instead of “the label of a vertex”. For example, “commit/open a vertex” means “commit/open the label of a vertex”.

4.1 Setup Phase

A prover can compute a Merkle commitment C that commits all vertices of $G_{n,k}$ using $2n + k$ space. The strategy is to pebble V_1 , compute Merkle commitment C_1 for V_1 , pebble V_2 , unpebble V_1 , compute Merkle commitment C_2 for V_2 , pebble V_3 , unpebble V_2 , and continue like this. C_1 to C_k are then committed into a single Merkle root C . The opening of a vertex v is the path from the root to the leaf corresponding to v in the Merkle tree.

After receiving the commitment C , V randomly selects l_0 vertices, and for each vertex v asks P to open v and, if v is not a source, also open all predecessors of v . Given the labels of all the predecessors (or if v is a source), V can then check if $h(v)$ is correctly computed. If any opening or $h(v)$ is incorrect, V rejects the commitment C and terminates the protocol. We note that P never stores the entire Merkle tree because it is too large. Thus, P has to pebble the graph for a

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

Such a probabilistic check ensures that the commitment C is “mostly correct”. If a label $h(v_i)$ under C is not correctly computed (as $h(x, i)$ or from v_i 's predecessors' labels under C), it's called a “fault”. If C contains m faults, V accepts C with $(1 - \frac{m}{nk})^{l_0}$ probability. If we want to ensure that $m < n/\delta$ with overwhelming probability, we can set $l_0 = (\ln 2)\delta k^2$; then for any $m \geq n/\delta$, C passes the probabilistic checking with less than 2^{-k} probability (k will be our security parameter).

A cheating prover is motivated to create faults using pseudorandom labels, because these faulty labels are essentially free pebbles that are always on the graph but take no space. Dziembowski et al. [4] 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. As we will see, this translates to a small loss in the soundness guarantee.

Finally, we would like to remark that this setup phase alone is a PoTS protocol. The proof ϕ is the openings of all the l_0 vertices and their predecessors. To pass the probabilistic check with non-negligible probability, P needs to pebble at least $n - n/\delta$ sinks ($> n/4$ with a proper δ), which requires at least $n/8 - n/\delta$ space or exponential time. The space gap is therefore at most $(2n + k)/(n/8 - n/\delta)$.

4.2 Audit Phase

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. The advice size is thus $2n$. Any vertices in V_i for $i < k$ are no longer needed. P and V now can also discard C and use C_k which commits V_k from this point onward.

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 $m < n/\delta$ 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 one last gap between Lemma 2 and what we need. Lemma 2 says any subset of $n/4$ sinks are hard to pebble, but we would hope to challenge P on $l_1 \ll n/4$ sinks. Therefore, we need to show that a majority of sinks are also hard to pebble individually.

Lemma 3. *For any initial configuration P_0 with $|P_0| < n/16$, at most $n/4$ sinks of $G_{n,k}$ can be pebbled individually using $n/16$ pebbles in 2^{k-2} moves starting from P_0 .*

Proof. Suppose for contradiction that there are more than $n/4$ 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 $n/4$ sinks, starting with $|P_0| < n/16 < n/8$, using at most $n/16 + n/16 = n/8$ pebbles in at most $2^{k-2}n/4 = 2^{k-1}n/8$ moves. This contradicts Lemma 2. \square

With more than three quarters of sinks being hard to pebble individually, we can set $l_1 = k/2$. A hard-to-pebble sink will be included in the challenge with at least $1 - 2^{-k}$ probability. Lastly, accounting for faults, no computationally bounded P using $n' = n/16 - n/\delta$ advice and space can pass the audit, making the advice gap $2n/n' < \frac{32}{1-16/\delta}$.

Table 1: Efficiency and space/advice gap of the PoPS protocol. The row for the setup phase represents the PoTS protocol. $l_0 = (\ln 2)\delta k^2$, $l_1 = k/2$, $d = 16$.

	message size	prover runtime	verifier runtime	N	N'
setup	$l_0(d + 1) \log_2(nk)$	$4nk$	$l_0(d + 1) \log_2(nk)$	$2n + k$	$n/8 - n/\delta$
audit	$l_1 \log_2 n$	$l_1 \log_2 n$	$l_1 \log_2 n$	$2n$	$n/16 - n/\delta$

4.3 Efficiency and Space/Advice Gap

Table 1 summarizes the efficiency and the space/advice gap of our PoPS and PoTS protocols. Message size is measured in label size (output size of \mathcal{H}). Runtime is measured by the number of calls to \mathcal{H} . Calls to \mathcal{H} have either 2 or 16 inputs, but we assume both take a unit amount of time.

An honest prover uses N space and advice. A computationally bounded cheating prover using less than N' space and advice has three chances to circumvent the exponential bound on pebbling moves: (1) passes the check with a commitment C containing $m \geq n/\delta$ faults, (2) gets lucky that all l_1 sinks in a challenge are easy to pebble under P_0 , or (3) guesses the label of at least one hard-to-pebble sink. Each of these happens with less than 2^{-k} probability. Thus, a cheating prover using less than N' space and advice succeeds with at most $3 \cdot 2^{-k}$ probability or has to run for over 2^{k-2} time. $k - 2$ can be viewed as the exact security parameter.

References

- [1] 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.
- [2] 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.
- [3] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology, CRYPTO92*, pages 139–147. Springer, 1992.
- [4] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Advances in Cryptology, CRYPTO 2015*, pages 585–605. Springer, 2015.
- [5] 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.
- [6] Nikolaos P. Karvelas and Aggelos Kiayias. Efficient proofs of secure erasure. In *Security and Cryptography for Networks*, pages 520–537. Springer, 2014.
- [7] 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.
- [8] Wolfgang J. Paul and Robert E. Tarjan. Time-space trade-offs in a pebble game. *Acta Informatica*, 10(2):111–115, 1978.
- [9] 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.
- [10] Daniele Perito and Gene Tsudik. Secure code update for embedded devices via proofs of secure erasure. In *ESORICS*, pages 643–662. Springer, 2010.
- [11] Herbert Robbins. A remark on Stirling’s formula. *The American Mathematical Monthly*, 62(1):26–29, 1955.