Proofs of Space

Stefan Dziembowski[1⋆], Sebastian Faust[2], Vladimir Kolmogorov[3⋆⋆], and Krzysztof Pietrzak[3⋆⋆⋆]

[1] University of Warsaw
[2] Ruhr-University Bochum
[3] IST Austria

Abstract. Proofs of work (PoW) have been suggested by Dwork and Naor (Crypto'92) as protection to a shared resource. The basic idea is to ask the service requestor to dedicate some non-trivial amount of computational work to every request. The original applications included prevention of spam and protection against denial of service attacks. More recently, PoWs have been used to prevent double spending in the Bitcoin digital currency system.

In this work, we put forward an alternative concept for PoWs – so-called *proofs of space* (PoS), where a service requestor must dedicate a significant amount of disk space as opposed to computation. We construct secure PoS schemes in the random oracle model (with one additional mild assumption required for the proof to go through), using graphs with high "pebbling complexity" and Merkle hash-trees. We discuss some applications, including follow-up work where a decentralized digital currency scheme called Spacecoin is constructed that uses PoS (instead of wasteful PoW like in Bitcoin) to prevent double spending.

The main technical contribution of this work is the construction of (directed, loop-free) graphs on N vertices with in-degree $O(\log\log N)$ such that even if one places $\Theta(N)$ pebbles on the nodes of the graph, there's a constant fraction of nodes that needs $\Theta(N)$ steps to be pebbled (where in every step one can put a pebble on a node if all its parents have a pebble).

1 Introduction

Proofs of Work (PoW). Dwork and Naor [20] suggested "proofs of work" (PoW) to address the problem of junk emails (aka. Spam). The basic idea is to require that an email be accompanied with some value related to that email that is moderately hard to compute but which can be verified very efficiently. Such a proof could for example be a value σ such that the hash value $\mathcal{H}(\mathsf{Email},\sigma)$ starts with t zeros. If we model the hash function $\mathcal{H}$ as a random oracle [12], then the sender must compute an expected 2^t hashes until she finds such a σ.[4] A useful property of this PoW is that there is no speedup when one has to find many proofs, i.e., finding s proofs requires $s2^t$ evaluations. The value t should be chosen such that it is not much of a burden for a party sending out a few emails per day (say, it takes 10 seconds to compute), but is expensive for a Spammer trying to send millions of messages. Verification on the other hand is extremely efficient, the receiver will accept σ as a PoW for Email, if the hash $\mathcal{H}(\mathsf{Email},\sigma)$ starts with t zeros, i.e., it requires only one evaluation of the hash funciton. PoWs have many applications, and are in particular used to prevent double spending in the Bitcoin digital currency system [42] which has become widely popular by now.

Despite many great applications, PoWs suffer from certain drawbacks. Firstly, running PoW costs energy – especially if they are used on a massive scale, like in the Bitcoin system. For this

⋆ Supported by the Foundation for Polish Science WELCOME/2010-4/2 grant founded within the framework of the EU Innovative Economy (National Cohesion Strategy) Operational Programme.

⋆⋆ VK is supported by European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no 616160.

⋆⋆⋆ Research supported by ERC starting grant (259668-PSPC).

[4] The hashed Email should also contain the receiver of the email, and maybe also a timestamp, so that the sender has to search for a fresh σ for each receiver, and also when resending the email at a later point in time.

reason Bitcoin has even been labelled an "environmental disaster" [5]. Secondly, by using dedicated hardware instead of a general purpose processor, one can solve a PoW at a tiny fraction of the hardware and energy costs, thus choosing "appropriate" parameters is tricky.

Proofs of Space (PoS). From a more abstract point of view, a proof of work is simply a means of showing that one invested a non-trivial amount of effort related to some statement. This general principle also works with resources other than computation like real money in micropayment systems [41] or human attention in CAPTCHAs [54, 16]. In this paper we put forward the concept of *proofs of space* where the resource in question is disk space.

PoS are partially motivated by the observation that users often have significant amounts of free disk anyway, and in this case using a PoS is essentially for free. This is in contrast to a PoW, as computing will always require energy consumption even if one contributes only CPU time of processors that would otherwise be idle.

A PoS is a protocol between a prover P and a verifier V which has two distinct phases. After an initialisation phase P is supposed to store some data $\mathcal{F}$ of size N, whereas V only holds some small piece of information. At any later time point V can initialise a proof execution phase, and at the end V outputs reject or accept. We require that V is highly efficient in both phases, whereas P is highly efficient in the execution phase providing he stored and has random access to the data $\mathcal{F}$.

As an illustrative application for a PoS, suppose that the verifier V is an organization that offers a free email service (we outline another example related to online polls in Appendix B.) To prevent that someone registers a huge number of fake-addresses for spamming, V might require users to dedicate some nontrivial amount of disk space, say 100GB, for every address registered. Occasionally, V will run a PoS to verify that the user really dedicates this space.

The simplest solution would be for the verifier V to generate a pseudorandom 100GB file $\mathcal{F}$ and send it to the prover P during an initialization phase. Later, V can ask P to send back some bits of $\mathcal{F}$ at random positions, making sure V stores (at least a large fraction of) $\mathcal{F}$. Unfortunately, with this solution, V still has to send a huge 100GB file to P, which makes this approach pretty much useless in practice.

We want a PoS where the computation, storage requirement and communication complexity of the verifier V during initialization and execution of the PoS is very small, in particular, at most polylogarithmic in the storage requirement N of the prover P and polynomial in some security parameter γ. In order to achieve small communication complexity, we must let the prover P generate a large file $\mathcal{F}$ locally during an initialization phase, which takes some time I. Note that I must be at least linear in N, our constructions will basically[5] achieve this lower bound. Later, P and V can run executions of the PoS which will be very cheap for V, and also for P, assuming the latter has stored $\mathcal{F}$.

Unfortunately, unlike in the trivial solution (where P sends $\mathcal{F}$ to V), now there is no way we can force a potentially cheating prover $\tilde{\mathsf{P}}$ to store $\mathcal{F}$ in-between the initialization and the execution of the PoS: $\tilde{\mathsf{P}}$ can delete $\mathcal{F}$ after initialization, and instead only store the (short) communication with V during the initialization phase. Later, before an execution of the PoS, P reconstructs $\mathcal{F}$ (in time I), runs the PoS, and deletes $\mathcal{F}$ once it is done.

[5] One of our constructions will achieve the optimal $I = \Theta(N)$ bound, our second construction achieves $I = O(N \log\log N)$.

We will thus consider a security definition where one requires that a cheating prover $\tilde{\mathsf{P}}$ can only make V accept with non-negligible probability if $\tilde{\mathsf{P}}$ either uses N_0 bits of storage in-between executions of the PoS *or* if $\tilde{\mathsf{P}}$ invests time T for every execution. Here $N_0 \leq N$ and $T \leq I$ are parameters, and ideally we want them to be not much smaller than N and I, respectively. Our actual security definition in Sect. 2 is more fine-grained, and besides the storage N_0 that $\tilde{\mathsf{P}}$ uses in-between initialization and execution, we also consider a bound N_1 on the total storage used by $\tilde{\mathsf{P}}$ during execution (including N_0, so $N_1 \geq N_0$).

High Level Description of our Scheme. We described above why the simple idea of having V sending a large (pseudorandom) file $\mathcal{F}$ to P does not give a PoS as the communication complexity is too large. Another simple idea that comes to mind is to have the V send the P a short description of a "randomly behaving" permutation $\pi : \{0,1\}^n \to \{0,1\}^n$. P then stores a table of $N = n2^n$ bits where the entry at position i is $\pi^{-1}(i)$. During the execution phase, V asks for the preimage of a random value y, which P can efficiently answer by returning the value $\pi^{-1}(y)$ at position y in the table. Unfortunately this scheme is no a good PoS because of time-memory trade-offs for inverting random functions [31] which allow to invert a random permutation over N values using only $\sqrt{N}$ time and space. For random functions (as opposed to permutations), it's still possible to invert in time and space $N^{2/3}$. We provide a more detailed discussion on this issue in Appendix A.

The actual PoS scheme we propose is based on hard to pebble graphs. During the initalisation phase, V sends the description of a hash function to P, who then labels the nodes of a hard to pebble graph using this function. Here the label of a node is computed as the hash of the labels of its children. V then computes a Merkle hash of all the labels, and sends this value to P. In the proof execution phase, V simply asks the P to open labels corresponding to some randomly chosen nodes.

Outline and our contribution. In this paper we introduce the concept of a PoS, which we formally define in Sect. 2. In Sect. 3 we discuss and motivate the model in which we prove our constructions secure (It is basically the random oracle model, but with an additional assumption). In Sect. 4 we explain how to reduce the security of a simple PoS (with an inefficient verifier) to a graph pebbling game. In Sect. 5 we show how to use hash-trees to make the verifier in the PoS from Sect. 4 efficient. In Sect. 6 we define our final construction and prove its security in Sect. 6.1 and Sect. 6.4.

Our proof uses a standard technique for proving lower bounds on the space complexity of computational problems, called *pebbling*. Typically, the lower bounds shown using this method are obtained via the *pebbling games* played on a directed graph. During the game a player can place pebbles on some vertices. The game starts with some pebbles already on the graph. Informally, placing a pebble on a vertex v corresponds to the fact that an algorithm keeps the label of a vertex v in his memory. Removing a pebble from a vertex corresponds therefore to deleting the vertex label from the memory. A pebble can be placed on a vertex v only if the vertices in-going to v have pebbles, which corresponds to the fact that computing v's label is possible only if the algorithm keeps in his memory the labels of the in-going vertices (in our case this will be achieved by defining the label of v to be a hash of the labels of its in-going vertices). The goal of the player is to pebble a certain vertex of the graph. This technique was used in cryptography already before [21–23]. For an introduction to the graph pebbling see, e.g., [48].

In Sect. 6.1 we consider two different (infinite families of) graphs with different (and incomparable) pebbling complexities. These graphs then also give PoS schemes with different parameters (cf. Theorem 3). Informally, the construction given in Theorem 1 proves a $\Omega(N/\log N)$ bound on the storage required by a malicious prover. Moreover, no matter how much time he is willing to spend during the execution of the protocol, he is forced to use at least $\Omega(N/\log N)$ storage when executing the protocol. Our second construction from Theorem 2 gives a stronger bound on the storage. In particular, a successful malicious prover either has to dedicate $\Theta(N)$ storage (i.e., almost as much as the N stored by the honest prover) or otherwise it has to use $\Theta(N)$ time with every execution of the PoS (after the initialization is completed). The second construction, whose proof appears in Appendix C and is based on superconcentrators, random bipartite expander graphs and on the graphs of Erdös, Graham and Szemerédi [24] is quite involved and is the main technical contribution of our paper.

We also note that a simple PoS can be constructed by storing the outputs of a random function in a sorted list. Unfortunately, this PoS achieves very weak security due to time-memory trade-offs as we will explain in Appendix A.

More related work and applications. Dwork and Naor [20] pioneered the concept of proofs of work as easy-to-check proofs of computational efforts. More concretely, they proposed to use the CPU running time that is required to carry out the proof as a measure of computational effort. In [1] Abadi, Burrows, Manasse and Wobber observed that CPU speeds may differ significantly between different devices and proposed as an alternative measure the number of times the memory is accessed (i.e., the number of cache misses) in order to compute the proof. This approach was formalized and further improved in [19, 55, 21, 4], which use pebbling based techniques. Such memory-hard functions cannot be used as PoS as the memory required to compute and verify the function is the same for provers and verifiers. This is not a problem for memory-hard functions as the here the memory just has to be larger than the cache of a potential prover, whereas in a PoS the storage is the main resource, and will typically be in the range of terabytes.

Originally put forward to counteract spamming, PoWs have a vast number of different applications such as metering web-site access [25], countering denial-of-service attacks [34, 9] and many more [33]. An important application for PoWs are digital currencies, like the recent Bitcoin system [42], or earlier schemes like the Micromint system of Rivest and Shamir [46]. The concept of using bounded resources such as computing power or storage to counteract the so-called "Sybil Attack", i.e., misuse of services by fake identities, has already mentioned in the work of Douceur [18].

PoW are used in Bitcoin to prevent double spending: honest *miners* must constantly devote more computational power towards solving PoWs than a potential adversary who tries to double spend. This results in a gigantic waste of energy [5] devoted towards keeping Bitcoin secure, and thus also requires some strong form of incentive for the miners to provide this computational power.[6] Recently a decentralized cryptocurrency called Spacecoin [43] was proposed which uses PoS instead of PoW to prevent double spending. In particular, a miner in Spacecoin who wants to dedicate N bits of disk space towards mining must just run the PoS initialisation phase once, and

[6] There are two mechanisms to incentivise mining: miners who solve a PoW get some fixed reward, this is currently the main incentive, but Bitcoin specifies that this reward will decrease over time. A second mechanism are transactions fees.

after that mining is extremely cheap: the miner just runs the PoS execution phase, which means accessing the stored space at a few positions, every few minutes.

A large body of work investigates the concepts of *proofs of storage* and *proofs of retrievability* (cf. [28, 29, 13, 8, 35, 17] and many more). These are proof systems where a verifier sends a file $\mathcal{F}$ to a prover, and later the prover can convince the verifier that it really stored or received the file. As discussed above, proving that one stores a (random) file certainly shows that one dedicates space, but these proof systems are not good PoS because the verifier has to send at least $|\mathcal{F}|$ bits to the verifier, and hence does not satisfy our polylogarithmic bound on the communication complexity.

Proof of Secure Erasure (PoSE) are related to PoS. Informally, a PoSE allows a space restricted prover to convince a verifier that he has erased its memory of size N. PoSE were suggested by Perito and Tsudik [45], who also proposed a scheme where the verifier sends a random file of size N to the prover, who then answers with a hash of this file. Dziembowski, Kazana and Wichs used graph pebbling to give a scheme with small communication complexity (which moreover is independent of N), but large $\Omega(N^2)$ computation complexity (for prover and verifier). Concurrently, and independently of our work, Karvelas and Kiayias [36], and also Ateniese et al [7] construct PoSE using graphs with high pebbling complexity. Interestingly, their schemes are basically the scheme one gets when running the initialisation and execution phase of our PoS (as in eq.(7) in Theorem 3).[7] [36] and [7] give a security proof of their construction in the random oracle model, and do not make any additional assumptions as we do. The reason is that to prove that our "collapsed PoS" (as described above) is a PoSE it is sufficient to prove that a prover uses much space *either* during initialisation or during execution. This follows from a (by now fairly standard) "ex post facto" argument as originally used in [21]. We have to prove something much stronger, namely, that the prover needs much space (or at least time) in the execution phase, even if he makes an unbounded amount of queries in the initialisation phase (we will discuss this in more detail in Section 3.1). As described above, a PoS (to be precise, a PoS where the execution phase requires large space, not just time) implies a PoSE, but a PoSE does not imply a PoS, nor can it be used for any of the applications mentioned in this paper. The main use-case for PoSE we know of is the one put forward by Perito and Tsudik [45], namely, to verify that some device has erased its memory. A bit unfortunately, Ateniese et al. [7] chose to call the type of protocols they construct also "proofs of space" which led to some confusion in the past.

Finally, let us mention a recent beautiful paper [14] which introduces the concept of "catalytic space". They prove a surprising result showing that *using* and *erasing* space is not the same relative to some additional space that is filled with random bits and must be in its original state at the end of the computation (i.e., it's only used as a "catalyst"). Thus, relative to such catalytic space, proving that one has access to some space as in a PoS, and proving that one has erased it, like in PoSE, really can be different things.

[7] There are some differences, the bounds in [7] are somewhat worse as they use hard-to-pebble graphs with worse parameters, and [36] do not use a Merkle hash-tree to make the computation of the verifier independent of N.

2 Defining Proofs of Space

We denote with $(\mathsf{out_V}, \mathsf{out_P}) \leftarrow \langle \mathsf{V}(\mathsf{in_V}), \mathsf{P}(\mathsf{in_P})\rangle(\mathsf{in})$ the execution of an interactive protocol between two parties P and V on shared input in, local inputs[8] $\mathsf{in_P}$ and $\mathsf{in_V}$, and with local outputs $\mathsf{out_V}$ and $\mathsf{out_P}$, respectively. A proof of space (PoS) is given by a pair of interactive random access machines,[9] a prover P and a verifier V. These parties run the PoS protocol in two phases: a PoS initialization and a PoS execution as defined below. The protocols are executed with respect to some statement id, given as common input (e.g., an email address in the example from the previous section). The identifier id is only required to make sure that P cannot reuse the same space to execute PoS for different statements.

Initialization is an interactive protocol with shared inputs an identifier id, storage bound $N \in \mathbb{N}$ and potentially some other parameters, which we denote with $\mathsf{prm} = (\mathsf{id}, N, \ldots)$. The execution of the initialization is denoted by $(\Phi, S) \leftarrow \langle \mathsf{V}, \mathsf{P}\rangle(\mathsf{prm})$, where Φ is short and S is of size N. V can output the special symbol $\Phi = \perp$, which means that it aborts (this can only be the case if V interacts with a cheating prover).

Execution is an interactive protocol during which P and V have access to the values stored during the initialization phase. The prover P has no output, the verifier V either accepts or rejects.

$$(\{\mathsf{accept}, \mathsf{reject}\}, \emptyset) \leftarrow \langle \mathsf{V}(\Phi), \mathsf{P}(S)\rangle(\mathsf{prm})$$

In an honest execution the initialization is done once at the setup of the system, e.g., when the user registers with the email service, while the execution can be repeated very efficiently many times without requiring a large amount of computation.

To formally define a proof of space, we introduce the notion of a (N_0, N_1, T) (dishonest) prover $\tilde{\mathsf{P}}$. $\tilde{\mathsf{P}}$'s storage after the initiation phase is bounded by at most N_0, while during the execution phase its storage is bounded to N_1 and its running time is at most T (here $N_1 \geq N_0$ as the storage during execution contains at least the storage after initialization). We remark that $\tilde{\mathsf{P}}$'s storage and running time is unbounded during the the initialization phase (but, as just mentioned, only N_0 storage is available in-between the initialization and execution phase).

A protocol (P, V) as defined above is a (N_0, N_1, T)-*proof of space*, if it satisfies the properties of completeness, soundness and efficiency defined below.

Completeness: We will require that for any honest prover P:

$$\Pr[\mathsf{out} = \mathsf{accept}\ :\ (\Phi, S) \leftarrow \langle \mathsf{V}, \mathsf{P}\rangle(\mathsf{prm})\ ,\ (\mathsf{out}, \emptyset) \leftarrow \langle \mathsf{V}(\Phi), \mathsf{P}(S)\rangle(\mathsf{prm})] = 1.$$

Note that the probability above is exactly 1, and hence the completeness is perfect.

[8] We use the expression "local input/output" instead the usual "private input/output", because in our protocols no values will actually be secret. The reason to distinguish between the parties' inputs is only due to the fact that P's input will be very large, whereas we want V to use only small storage.

[9] In a PoS, we want the prover P to run in time much less than its storage size. For this reason, we must model our parties as random access machines (and not, say Turing machines), where accessing a storage location is assumed to take constant (or at most polylogarithmic) time.

Soundness: For any (N_0, N_1, T)-adversarial prover $\tilde{\mathsf{P}}$ the probability that V accepts is negligible in some statistical security parameter γ. More precisely, we have

$$\Pr[\mathsf{out} = \mathsf{accept} \ : \ (\Phi, S) \leftarrow \langle \mathsf{V}, \tilde{\mathsf{P}} \rangle(\mathsf{prm}), (\mathsf{out}, \emptyset) \leftarrow \langle \mathsf{V}(\Phi), \tilde{\mathsf{P}}(S) \rangle(\mathsf{prm})] \leq 2^{-\Theta(\gamma)} \quad (1)$$

The probability above is taken over the random choice of the public parameters prm and the coins of $\tilde{\mathsf{P}}$ and V.[10]

Efficiency: We require the verifier V to be efficient, by which (here and below) we mean at most polylogarithmic in N and polynomial in some security parameter γ. Prover P must be efficient during execution, but can run in time $\mathrm{poly}(N)$ during initialization.[11]

In the soundness definition above, we only consider the case where the PoS is executed only once. This is without loss of generality for PoS where V is stateless (apart from Φ) and holds no secret values, and moreover the honest prover P uses only read access to the storage of size N holding S. The protocols in this paper are of this form. We will sometimes say that a PoS is (N_0, N_1, T)-secure if it is a (N_0, N_1, T)-*proof of space*.

It is instructive to observe what level of security trivially cannot be achieved by a PoS. Below we use small letters n, t, c to denote values that are small, i.e., polylogarithmic in N and polynomial in a security parameter γ. If the honest prover P is an $(N, N+n, t)$ prover, where t, n denote the time and storage requirements of P during execution, then there exists *no*

1. $(N, N+n, t)$-PoS, as the honest prover "breaks" the scheme by definition, and
2. $(c, I+t+c, I+t)$-PoS, where c is the number of bits sent by V to P during initialization. To see this, consider a malicious prover $\tilde{\mathsf{P}}$ that runs the initialization like the honest P, but then only stores the messages sent by V during initialization instead of the entire large S. Later, during execution, $\tilde{\mathsf{P}}$ can simply emulate the initialization process (in time I) to get back S, and run the normal execution protocol (in time t).

3 The model

We analyze the security and efficiency of our PoS in the random oracle (RO) model [12], making an additional assumption on the behavior of adversaries, which we define and motivate below. Recall that in the RO model, we assume that all parties (including adversaries) have access to the same random function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^L$. In practice, one must instantiate $\mathcal{H}$ with a real hash function like SHA3. Although security proofs in the RO model are just a heuristic argument for real-world security, and there exist artificial schemes where this heuristic fails [15, 27, 38], the model has proven surprisingly robust in practice.

Throughout, we fix the output length of our random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^L$ to some $L \in \mathbb{N}$, which should be chosen large enough, so it is infeasible to find collisions. As finding a

[10] Our construction is based on a hash-function $\mathcal{H}$, which will be part of prm and we require to be collision resistant. As assuming collision resistance for a fixed function is not meaningful [47], we must either assume that the probability of Eq. (1) is over some distribution of identities id (which can then be used as a hash key), or, if we model $\mathcal{H}$ as a random oracle, over the choice of the random oracle.

[11] As explained in the introduction, P's running time I during initialization must be at least linear in the size N of the storage. Our construction basically match this $I = \Omega(N)$ lower bound as mentioned in Footnote 5.

collision requires roughly $2^{L/2}$ queries, setting $L = 512$ and assuming that the total number of oracle queries during the entire experiment is upper bounded by, say $2^{L/3}$, would be a conservative choice.

3.1 Modeling the malicious prover

In this paper, we want to make statements about adversaries (malicious provers) $\tilde{\mathsf{P}}$ with access to a random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^L$ and bounded by three parameters N_0, N_1, T. They run in two phases:

1. In a first (initialization) phase, $\tilde{\mathsf{P}}$ makes queries[12] $\mathcal{A} = (a_1, \ldots, a_q)$ to $\mathcal{H}$ (adaptively, i.e., a_i can be a function of $\mathcal{H}(a_1), \ldots, \mathcal{H}(a_{i-1})$). At the end of this phase, $\tilde{\mathsf{P}}$ stores a file S of size $N_0 L$ bits, and moreover he must commit to a subset of the queries $\mathcal{B} \subseteq \mathcal{A}$ of size N (technically, we'll do this by a Merkle hash-tree).
2. In a second phase, $\tilde{\mathsf{P}}(S)$ is asked to output $\mathcal{H}(b)$ for some random $b \in \mathcal{B}$. The malicious prover $\tilde{\mathsf{P}}(S)$ is allowed a total number T of oracle queries in this phase, and can use up to $N_1 L$ bits of storage (including the $N_0 L$ bits for S).

As $\mathcal{H}$ is a random oracle, one cannot compress its uniformly random outputs. In particular, as S is of size $N_0 L$, it cannot encode more than N_0 outputs of $\mathcal{H}$. We will make the simplifying assumption that we can explicitly state which outputs these are by letting $S_{\mathcal{H}} \subset \{0,1\}^L, |S_{\mathcal{H}}| \leq N_0$ denote the set of all possible outputs $\mathcal{H}(a), a \in \mathcal{A}$ that $\tilde{\mathsf{P}}(S)$ can write down during the second phase without explicitly querying $\mathcal{H}$ on input a in the 2nd phase.[13] Similarly, the storage bound $N_1 L$ during execution implies that $\tilde{\mathsf{P}}$ cannot store more than N_1 outputs of $\mathcal{H}$ at any particular time point, and we assume that this set of $\leq N_1$ inputs is well defined at any time-point. The above assumption will allow us to bound the advantage of a malicious prover in terms of a pebbling game.

The fact that we need the additional assumption outlined above and cannot reduce the security of our scheme to the plain random oracle model is a bit unsatisfactory, but unfortunately the standard tools (in particular, the elegant "ex post facto" argument from [21]), typically used to reduce pebbling complexity to the number of random oracle queries, cannot be applied in our setting due to the auxiliary information about the random oracle the adversary can store. We believe that a proof exploiting the fact that random oracles are incompressible using techniques developed in [50, 30] can be used to avoid this additional assumption, and we leave this question as interesting future work.

[12] The number q of queries in this phase is unbounded, except for the huge exponential $2^{L/3}$ bound on the total number of oracle queries made during the entire experiment by all parties mentioned above.

[13] Let us stress that we do not claim that such an $S_{\mathcal{H}}$ exists for every $\tilde{\mathsf{P}}$, one can easily come up with a prover where this is not the case (as we will show below). All we need is that for every (N_0, N_1, T) prover $\tilde{\mathsf{P}}$, there exists another prover $\tilde{\mathsf{P}}'$ with (almost) the same parameters and advantage, that obeys our assumption.

An adversary with $N_0 = N_1 = T = 1$ not obeying our assumption is, e.g., a $\tilde{\mathsf{P}}$ that makes queries 0 and 1 and stores $S = \mathcal{H}(0) \oplus \mathcal{H}(1)$ in the first phase. In the second phase, $\tilde{\mathsf{P}}(S)$ picks a random $b \leftarrow \{0,1\}$, makes the query b, and can write down $\mathcal{H}(b), \mathcal{H}(1-b) = S \oplus \mathcal{H}(b)$. Thus, $\tilde{\mathsf{P}}(S)$ can write $2 > N_0 = 1$ values $\mathcal{H}(0)$ or $\mathcal{H}(1)$ without quering them in the 2nd phase.

3.2 Storage and time complexity

Time complexity. Throughout, we let the *running time* of honest and adversarial parties be the *number of oracle queries* they make. We also take into account that hashing long messages is more expensive by "charging" k queries for a single query on an input of bit-length $L(k-1)+1$ to Lk. Just counting oracle queries is justified by the fact that almost all computation done by honest parties consists of invocations of the random-oracle, thus we do not ignore any computation here. Moreover, ignoring any computation done by adversaries only makes the security proof stronger.

Storage complexity. Unless mentioned otherwise, the storage of honest and adversarial parties is measured by the number of outputs $y = \mathcal{H}(x)$ stored. The honest prover P will only store such values by construction; for malicious provers $\tilde{\mathsf{P}}$ this number is well defined under the assumption from Sect. 3.1.

4 PoS from graphs with high pebbling complexity

The first ingredient of our proof uses graph pebbling. We consider a directed, acyclic graph $G = (V, E)$. The graph has $|V| = N$ vertices, which we label with numbers from the set $[N] = \{1, \ldots, N\}$. With every vertex $v \in V$ we associate a value $w(v) \in \{0,1\}^L$, and extend the domain of w to include also ordered tuples of elements from V in the following way: for $V' = (v_1, \ldots, v_n)$ (where $v_i \in V$) we define $w(V') = (w(v_1), \ldots, w(v_n))$. Let $\pi(v) = \{v' : (v', v) \in E\}$ denote v's predecessors (in some arbitrary, but fixed order). The value $w(v)$ of v is computed by applying the random oracle to the index v and the values of its predecessors

$$w(v) = \mathcal{H}(v, w(\pi(v))) \; . \tag{2}$$

Note that if v is a source, i.e., $\pi(v) = \emptyset$, then $w(v)$ is simply $\mathcal{H}(v)$. Our PoS will be an extension of the simple basic PoS $(\mathsf{P}_0, \mathsf{V}_0)[G, \Lambda]$ from Figure 1, where Λ is an efficiently samplable distribution that outputs a subset of the vertices V of $G = (V, E)$. This PoS does not yet satisfy the efficiency requirement from Sect. 2, as the complexity of the verifier needs to be as high as the one of the prover. This is because, in order to perform the check in Step 3 of the execution phase, the verifier needs to compute $w(C)$ himself. In our model, as discussed in Sect. 3.1, the only way a

> **Parameters** $\mathsf{prm} = (\mathsf{id}, N, G = (V, E), \Lambda)$, where G is a graph on $|V| = N$ vertices and Λ is an efficiently samplable distribution over V^β (we postpone specifying β as well as the function of id to Sect. 6).
> **Initialization** $(S, \emptyset) \leftarrow \langle \mathsf{P}_0, \mathsf{V}_0 \rangle(\mathsf{prm})$ where $S = w(V)$.
> **Execution** $(\mathsf{accept/reject}, \emptyset) \leftarrow \langle \mathsf{V}(\emptyset), \mathsf{P}(S) \rangle(\mathsf{prm})$
> 1. $\mathsf{V}_0(\emptyset)$ samples $C \leftarrow \Lambda$ and sends C to P_0.
> 2. $\mathsf{P}_0(S)$ answers with $A = w(C) \subset S$.
> 3. $\mathsf{V}_0(\emptyset)$ outputs accept if $A = w(C)$ and reject otherwise.

Fig. 1. The basic PoS $(\mathsf{P}_0, \mathsf{V}_0)[G, \Lambda]$ (with inefficient verifier V_0).

malicious prover $\tilde{\mathsf{P}}_0(S)$ can determine $w(v)$ is if $w(v) \in S_{\mathcal{H}}$ is in the encoded set of size at most

N_0, or otherwise by explicitly making the oracle query $\mathcal{H}(v, w(\pi(v)))$ during execution. Note that if $w(i) \notin S_{\mathcal{H}}$ for some $i \in \pi(v)$, then $\tilde{\mathsf{P}}_0(S)$ will have to make even more queries recursively to learn $w(v)$. Hence, in order to prove (N_0, N_1, T)-security of the PoS $(\mathsf{P}_0, \mathsf{V}_0)[G, \Lambda]$ in our idealized model, it suffices to upper bound the advantage of Player 1 in the following pebbling game on $G = (V, E)$:

1. Player 1 puts up to N_0 initial pebbles on the vertices of V.
2. Player 2 samples a subset $C \leftarrow \Lambda$ of size α of challenge vertices.
3. Player 1 applies a sequence of up to T steps according to the following rules:
   (i) it can place a pebble on a vertex v if (1) all its predecessors $u \in \pi(v)$ are pebbled and (2) there are currently less than N_1 vertices pebbled.
   (ii) it can remove a pebble from any vertex.
4. Player 1 wins if it places pebbles on all vertices of C.

In the pebbling game above, Step 1 corresponds to a malicious prover $\tilde{\mathsf{P}}_0$ choosing the set $S_{\mathcal{H}}$. Step 3 corresponds to $\tilde{\mathsf{P}}_0$ computing values according to the rules in Eq. (2), while obeying the N_1 total storage bound. Putting a pebble corresponds to invoking $y = \mathcal{H}(x)$ and storing the value y. Removing a pebble corresponds to deleting some previously computed y.

5 Efficient verifiers using hash trees

The PoS described in the previous section does not yet meet our Definition from Sect. 2 as V_0 is not efficient. In this section we describe how to make the verifier efficient, using hash-trees, a standard cryptographic technique introduced by Ralph Merkle [39]

Using hash trees for committing. A hash-tree allows a party P to compute a commitment $\phi \in \{0,1\}^L$ to N data items $x_1, \ldots, x_N \in \{0,1\}^L$ using $N-1$ invocations of a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^L$. Later, P can prove to a party holding ϕ what the value of any x_i is, by sending only $L \log N$ bits. For example, for $N = 8$, P commits to $x_1, \ldots, x_N$ by hashing the x_i's in a tree like structure as

$$\phi = \mathcal{H}(\ \mathcal{H}(\ \mathcal{H}(x_1, x_2), \mathcal{H}(x_3, x_4)\ ), \mathcal{H}(\ \mathcal{H}(x_5, x_6), \mathcal{H}(x_7, x_8)\ )\ )$$

We will denote with $\mathcal{T}^{\mathcal{H}}(x_1, \ldots, x_N)$ the $2N-1$ values of all the nodes (including the N leaves x_i and the root ϕ) of the hash-tree, e.g., for $N = 8$, where we define $x_{ab} = \mathcal{H}(x_a, x_b)$

$$\mathcal{T}^{\mathcal{H}}(x_1, \ldots, x_8) = \{x_1, \ldots, x_8, x_{12}, x_{34}, x_{56}, x_{78}, x_{1234}, x_{5678}, \phi = x_{12345678}\}$$

The prover P, in order to later efficiently open any x_i, will store all $2N-1$ values $\mathcal{T} = \mathcal{T}^{\mathcal{H}}(x_1, \ldots, x_N)$, but only send the single root element ϕ to a verifier V. Later P can "open" any value x_i to V by sending x_i and the $\log N$ values, which correspond to the siblings of the nodes that lie on the path from x_i to ϕ, e.g., to open x_3 P sends x_3 and $\mathsf{open}(\mathcal{T}, 3) = (x_{12}, x_4, x_{5678})$ and the prover checks if

$$\mathsf{vrfy}(\phi, 3, x_3, (x_{12}, x_4, x_{5678})) = \Big(\mathcal{H}(x_{12}, \mathcal{H}(x_3, x_4)), x_{56789}) \stackrel{?}{=} \phi\Big)$$

As indicated above, we denote with $\mathsf{open}(\mathcal{T}, i) \subset \mathcal{T}$ the $\log N$ values P must send to V in order to open x_i, and denote with $\mathsf{vrfy}(\phi, i, x_i, o) \to \{\mathsf{accept}, \mathsf{reject}\}$ the above verification procedure. This scheme is correct, i.e., for $\phi, \mathcal{T}$ computed as above and any $i \in [N]$, $\mathsf{vrfy}(\phi, i, x_i, \mathsf{open}(\mathcal{T}, i)) = \mathsf{accept}$.

The security property provided by a hash-tree states that it is hard to open any committed value in more than one possible way. This "binding" property can be reduced to the collision resistance of $\mathcal{H}$: from any $\phi, i, (x, o), (x', o'), x \neq x'$ where $\mathsf{vrfy}(\phi, i, x, o) = \mathsf{vrfy}(\phi, i, x', o') = \mathsf{accept}$, one can efficiently extract a collision $z \neq z', \mathcal{H}(z) = \mathcal{H}(z')$ for $\mathcal{H}$.

to the graph based PoS from Figure 1, where the prover $\mathsf{P}(\mathsf{prm})$ commits to $x_1 = w(v_1), \ldots, x_N = w(v_N)$ by computing a hash tree $\mathcal{T} = \mathcal{T}^{\mathcal{H}}(x_1, \ldots, x_N)$ and sending its root ϕ to V. In the execution phase, the prover must then answer a challenge c not only with the value $x_c = w(c)$, but also open c by sending $(x_c, \mathsf{open}(\mathcal{T}, c))$ which P can do without any queries to $\mathcal{H}$ as it stored $\mathcal{T}$.

If a cheating prover $\tilde{\mathsf{P}}(\mathsf{prm})$ sends a correctly computed ϕ during the initialization phase, then during execution $\tilde{\mathsf{P}}(\mathsf{prm}, S)$ can only make $\mathsf{V}(\mathsf{prm}, \phi)$ accept by either answering each challenge c with the correct value $w(c)$, or by breaking the binding property of the hash-tree (and thus the collision resistance of the underlying hash-function).

We are left with the challenge to deal with a prover who might cheat and send a wrongly computed $\tilde{\Phi} \neq \phi$ during initialization. Some simple solutions are

- Have V compute ϕ herself. This is not possible as we want V's complexity to be only polylog in N.
- Let P prove, using a proof system like computationally sound (CS) proofs [40] or universal arguments [10], that ϕ was computed correctly. Although these proof systems do have polylogarithmic complexity for the verifier, and thus formally would meet our efficiency requirement, they rely on the PCP theorem and thus are not really practical.

Dealing with wrong commitments. Unless $\tilde{\mathsf{P}}$ breaks the collision resistance of $\mathcal{H}$, no matter what commitment $\tilde{\Phi}$ the prover P sends to V, he can later only open it to some fixed N values which we will denote $\tilde{x}_1, \ldots, \tilde{x}_N$.[14] We say that $\tilde{x}_i$ is consistent if

$$\tilde{x}_i = \mathcal{H}(i, \tilde{x}_{i_1}, \ldots, \tilde{x}_{i_d}) \text{ where } \pi(i) = \{i_1, \ldots, i_d\} \tag{3}$$

Note that if *all* $\tilde{x}_i$ are consistent, then $\tilde{\Phi} = \phi$. We add a second initialization phase to the PoS, where V will check the consistency of α random $\tilde{x}_i$'s. This can be done by having $\tilde{\mathsf{P}}$ open $\tilde{x}_i$ and $\tilde{x}_j$ for all $j \in \pi(i)$. If $\tilde{\mathsf{P}}$ passes this check, we can be sure that with high probability a large fraction of the $\tilde{x}_i$'s is consistent. More concretely, if the number of challenge vertices is $\alpha = \varepsilon t$ for some $\varepsilon > 0$, then $\tilde{\mathsf{P}}$ will fail the check with probability $1 - 2^{-\Theta(t)}$ if more than an ε-fraction of the $\tilde{x}_i$'s are inconsistent.

A cheating $\tilde{\mathsf{P}}$ might still pass this phase with high probability with an $\tilde{\Phi}$ where only $1 - \varepsilon$ fraction of the $\tilde{x}_i$ are consistent for some sufficiently small $\varepsilon > 0$. As the inconsistent $\tilde{x}_i$ are not outputs of $\mathcal{H}$, $\tilde{\mathsf{P}}$ can chose their value arbitrarily, e.g., all being 0^L. Now $\tilde{\mathsf{P}}$ does not have to store this εN inconsistent values $\tilde{x}_j$ while still knowing them.

[14] Potentially, $\tilde{\mathsf{P}}$ cannot open some values at all, but wlog. we assume that it can open every value in exactly one way.

In our idealized model as discussed in Sect. 3.1, one can show that this is already all the advantage $\tilde{\mathsf{P}}$ gets. We can model an εN fraction of inconsistent $\tilde{x}_i$'s by slightly augmenting the pebbling game from Sect. 4. Let the pebbles from the original game be *white* pebbles. We now additionally allow player 1 to put εN *red* pebbles (apart from the N_0 white pebbles) on V during step 1. These red pebbles correspond to inconsistent values. The remaining game remains the same, except that player 1 is never allowed to remove red pebbles.

We observe that being allowed to initially put an additional εN red pebbles is no more useful than getting an additional εN white pebbles (as white pebbles are strictly more useful because, unlike red pebbles, they later can be removed.) Translated back to our PoS, in order prove (N_0, N_1, T)-security of our PoS allowing up to εN inconsistent values, it suffices to prove $(N_0 - \varepsilon N, N_1 - \varepsilon N, T)$-security of the PoS, assuming that the initial commitment is computed honestly, and there are no inconsistent values (and thus no red pebbles in the corresponding game).

6 Our Main Construction

Below we formally define our PoS (P, V). The common input to P, V are the parameters $\mathsf{prm} = (\mathsf{id}, 2N, \gamma, G, \Lambda)$, which contain the identifier $\mathsf{id} \in \{0,1\}^*$, a storage bound $2N \in \mathbb{N}$ (i.e., $2NL$ bits),[15] a statistical security parameter γ, the description of a graph $G(V, E)$ on $|V| = N$ vertices and an efficiently samplable distribution Λ which outputs some "challenge" set $C \subset V$ of size $\alpha = \alpha(\gamma, N)$.

Below $\mathcal{H}$ denotes a hash function, that depends on id: given a hash function $\mathcal{H}'(.)$ (which we will model as a random oracle in the security proof), throughout we let $\mathcal{H}(.)$ denote $\mathcal{H}'(\mathsf{id}, .)$. The reason for this is simply so we can assume that the random oracles $\mathcal{H}'(\mathsf{id}, .)$ and $\mathcal{H}'(\mathsf{id}', .)$ used in PoS with different identifiers $\mathsf{id} \neq \mathsf{id}'$ are independent, and thus anything stored for the PoS with identifier id is useless to answer challenges in a PoS with different identifier id'.

Initialization $(\Phi, S) \leftarrow \langle \mathsf{V}, \mathsf{P} \rangle(\mathsf{prm})$:

1. P **sends** V **a commitment** ϕ **to** $w(V)$
   - P computes the values $x_i = w(i)$ for all $i \in V$ as in Eq. (2).
   - P's output is a hash-tree $S = \mathcal{T}^{\mathcal{H}}(x_1, \ldots, x_N)$, which requires $|S| = (2N-1)L$ bits) as described in Sect. 5.
   - P sends the root $\phi \in S$ to V.
2. P **proves consistency of** ϕ **for** $\alpha = \alpha(\gamma, N)$ **random values**
   - V picks a set of challenges $C \leftarrow \Lambda$ where the size of C is α and sends C to P.
   - For all $c \in C$, P opens the value corresponding to c and all its predecessors to V by sending, for all $c \in C$

$$\{(x_i, \mathsf{open}(S, i)) \; : \; i \in \{c, \pi(c)\}\}$$

[15] We set the bound to $2N$, so if we let N denote the number of vertices in the underlying graph, we must store $2N-1$ values of the hash-tree.

– V verifies that P sends all the required openings, and they are consistent, i.e., for all $c \in C$ the opened values $\tilde{x}_c$ and $\tilde{x}_i, i \in \pi(c) = (i_1, \ldots, i_d)$ must satisfy $\tilde{x}_c = \mathcal{H}(c, \tilde{x}_{i_1}, \ldots, \tilde{x}_{i_d})$, and the verification of the opened commitments passes. If either check fails, V outputs $\Phi = \perp$ and aborts. Otherwise, V outputs $\Phi = \phi$, and the initialization phase is over.

Execution $(\mathsf{accept}/\mathsf{reject}, \emptyset) \leftarrow \langle \mathsf{V}(\Phi), \mathsf{P}(S)\rangle(\mathsf{prm})$**:**

P **proves it stores the committed values by opening a random** $\beta = \Theta(\gamma)$ **subset of them**

– V picks a challenge set $C \subset V$ of size $|C| = \beta$ at random, and sends C to P.
– P answers with $\{o_c = (x_c, \mathsf{open}(S, c))\ :\ c \in C\}$.
– V checks for every $c \in C$ if $\mathsf{vrfy}(\Phi, c, o_c) \stackrel{?}{=} \mathsf{accept}$. V outputs accept if this is the case and reject otherwise.

6.1 Constructions of the graphs

We consider the following pebbling game, between a player and a challenger, for a directed acyclic graph $G = (V, E)$ and a distribution λ over V.

1. Player puts initial pebbles on some subset $U \subseteq V$ of vertices.
2. Challenger samples a "challenge vertex" $c \in V$ according to λ.
3. Player applies a sequence of steps according to the following rules:
   (i) it can place a pebble on a vertex v if all its predecessors $u \in \pi(v)$ are pebbled.
   (ii) it can remove a pebble from any vertex.
4. Player wins if it places a pebble on c.

Let $S_0 = |U|$ be the number of initial pebbles, S_1 be the total number of used pebbles (or equivalently, the maximum number of pebbles that are present in the graph at any time instance, including initialization), and let T be the number of pebbling steps given in 3i). The definition implies that $S_1 \geq S_0$ and $T \geq S_1 - S_0$. Note, with $S_0 = |V|$ pebbles the player can always achieve time $T = 0$: it can just place initial pebbles on V.

Definition 1. *Consider functions $f = f(N, S_0)$ and $g = g(N, S_0, S_1)$. A family of graphs $\{G_N = (V_N, E_N) \mid |V_N| = N \in \mathbb{N}\}$ is said to have* pebbling complexity $\Omega(f, g)$ *if there exist constants $c_1, c_2, \delta > 0$ and distributions λ_N over V_N such that for any player that wins the pebbling game on (G_N, λ_N) (as described above) with probability 1 it holds that*

$$\Pr[\ S_1 \geq c_1 f(N, S_0)\ \wedge\ T \geq c_2 g(N, S_0, S_1)\ ] \geq \delta \tag{4}$$

Let $\mathcal{G}(N, d)$ be the set of directed acyclic graphs $G = (V, E)$ with $|V| = N$ vertices and the maximum in-degree at most d. We now state our two main pebbling theorems:

Theorem 1. *There exists an explicit family of graphs $G_N \in \mathcal{G}(N, 2)$ with pebbling complexity*

$$\Omega(N/\log N, 0) \tag{5}$$

In the next theorem we use the *Iverson bracket* notation: $[\phi] = 1$ if statement ϕ is true, and $[\phi] = 0$ otherwise.

Theorem 2. *There exists a family of graphs $G_N \in \mathcal{G}(N, O(\log\log N))$ with pebbling complexity*

$$\Omega(0, [S_0 < \tau N] \cdot \max\{N, N^2/S_1\}) \tag{6}$$

for some constant $\tau \in (0,1)$. It can be constructed by a randomized algorithm with a polynomial expected running time that produces the desired graph with probability at least $1 - 2^{-\Theta(N/\log N)}$.

Remark 1. As shown in [32], any graph $G \in \mathcal{G}(N, O(1))$ can be entirely pebbled using $S_1 = O(N/\log N)$ pebbles (without any initial pebbles). This implies that expression $N/\log N$ in Theorem 1 cannot be improved upon. Note, this still leaves the possibility of a graph that can be pebbled using $O(N/\log N)$ pebbles only with a large time T (e.g. superpolynomial in N). Examples of such graph for a non-interactive version of the pebble game can be found in [37]. Results stated in [37], however, do not immediately imply a similar bound for our interactive game.

6.2 Proof of Theorem 1

Paul, Tarjan and Celoni [44] presented a family of graphs $G_{(i)}$ for $i = 8, 9, 10, \ldots$ with m_i sources, m_i sinks and n_i nodes where $m_i = 2^i$ and $n_i = \Theta(i2^i)$. The following claim is a special case of their Lemma 2.

Lemma 1. *For any initial configuration of no more than cm_i pebbled vertices (with $c = 1/256$) there exists a sink whose pebbling requires a time instance at which at least cm_i pebbles are on the graph.*

We can show the following.

Corollary 1. *For a subset $U \subseteq V$ let X_U be the set of sinks whose pebbling requires at least $\frac{1}{2}cm_i$ pebbles starting with U as the initial set of pebbles. If $|U| \le \frac{1}{2}cm_i$ then $|X_U| \ge \frac{1}{2}cm_i$.*

Proof. Assume that $|U| \le \frac{1}{2}cm_i$ and $|X_U| < \frac{1}{2}cm_i$ for some $U \subseteq V$. Consider the following pebbling algorithm. First, place initial pebbles at vertices in $U \cup X_U$. To pebble remaining sinks $v \notin X_U$, go through them in some order and do the following:

1. Remove all pebbles except those in U.
2. By definition of X_U, vertex v can be pebbled using fewer than $\frac{1}{2}cm_i$ pebbles. Run a modification of the corresponding algorithm where pebbles in U are never removed.

This algorithm pebbles all sinks in some order, starts with $|U| + |X_U| < cm_i$ initial pebbles, and uses fewer than $|U| + \frac{1}{2}cm_i \le cm_i$ pebbles at each time instance. By Lemma 1, this is a contradiction.

□

We can now prove Theorem 1. Consider $N \ge n_1$, and let i be the largest integer such that $n_i \le N$. Let G_N be the graph obtained from $G_{(i)}$ by adding $N - n_i$ “dummy” vertices. It can be checked that $m_i = \Theta(N/\log N)$.

Let $\tilde{V}$ be the set of outputs of G_N excluding dummy vertices, with $|\tilde{V}| = m_i$. We define λ to be the uniform probability distribution over vertices $c \in \tilde{V}$.

Let us show that $S_1 \geq \frac{1}{2}cm_i = \Theta(N/\log N)$ with probability at least $\delta = \frac{1}{2}c$. Assume that $|U| = S_0 \leq \frac{1}{2}cm_i$, otherwise the claim is trivial. By Corollary 1 we have $|X_U| \geq \frac{1}{2}c|\tilde{V}|$. Using the definition of set X_U, we get

$$Pr[S_1 \geq \tfrac{1}{2}cm_i] \geq Pr[c \in X_U] = |X_U|/|\tilde{V}| \geq \tfrac{1}{2}c = \delta.$$

6.3 Sketch of the proof of Theorem 2

The complete proof of this theorem is given in Appendix C; here we give a brief summary of our techniques. We use a new construction which relies on three building blocks: (i) random bipartite graphs $R^d_{(m)} \in \mathcal{G}(2m, d)$ with m inputs and m outputs; (ii) superconcentrator graphs $C_{(m)}$ with m inputs and m outputs; (iii) graphs $D_t = ([t], E_t)$ of Erdös, Graham and Szemerédi [24] with *dense long paths*. These are directed acyclic graphs with t vertices and $\Theta(t \log t)$ edges (of the form (i, j) with $i < j$) that satisfy the following for some constant $\eta \in (0, 1)$ and a sufficiently large t: for any subset $X \subseteq [t]$ of size at most ηt graph D_t contains a path of length at least ηt that avoids X. We show that family D_t can be chosen so that the maximum in-degree is $\Theta(\log t)$. The main component of our construction is graph $\tilde{G}^d_{(m,t)}$ defined as follows:

- Add mt nodes $\tilde{V} = V_1 \cup \ldots \cup V_t$ where $|V_1| = \ldots = |V_t| = m$. This will be the set of challenges.
- For each edge (i, j) of graph D_t add a copy of graph $R^d_{(m)}$ from V_i to V_j, i.e. identify the inputs of $R^d_{(m)}$ with nodes in V_i (using an arbitrary permutation) and the outputs of $R^d_{(m)}$ with nodes in V_j (again, using an arbitrary permutation).

We set $d = \Theta(1)$, $t = \Theta(\log N)$ and $m = \Theta(N/t)$ (with specific constants), then $\tilde{G}^d_{(m,t)} \in \mathcal{G}(mt, O(\log\log N))$.

Note that a somewhat similar graph was used by Dwork, Naor and Wee [21]. They connect bipartite graphs $R^d_{(m)}$ consecutively, i.e. instead of graph D_t they use a chain graph with t nodes. Dwork et al. give an intuitive argument that removing at most τm nodes from each layer $V_1, \ldots, V_t$ (for some constant $\tau < 1$) always leaves a graph which is "well-connected": informally speaking, many nodes of V_1 are still connected to many nodes of V_t. (We formalize their argument in Appendix C.) However, this does not hold if more than $m = \Theta(N/\log N)$ nodes are allowed to be removed: by placing initial pebbles on, say, the middle layer $V_{t/2}$ one can completely disconnect V_1 from V_t.

In contrast, in our construction removing any $\tau' N$ nodes still leaves a graph which is "well-connected". Our argument is as follows. If constant τ' is sufficiently small then there can be at most ηt layers with more than τm initial pebbles (for a given constant $\tau < 1$). By the property of D_t, there exists a sufficiently long path P in D_t that avoids those layers. We can thus use the argument above for the subgraph corresponding to P. We split P into three parts of equal size, and show that many nodes in the first part are connected to many nodes in the third part.

In this way we prove that graphs $\tilde{G}^d_{(m,t)}$ have pebbling complexity $\Omega(0, [S_0 < \tau N] \cdot N)$. To get complexity $\Omega(0, [S_0 < \tau N] \cdot \max\{N, N^2/S_1\})$, we add mt extra nodes V_0 and a copy of superconcentrator $C_{(mt)}$ from V_0 to $\tilde{V}$. We then use a standard "basic lower bound argument" for superconcentrators [37].

6.4 Putting things together

Combining the results and definitions from the previous sections, we can now state our main theorem.

Theorem 3. *In the model from Sect. 3.1, for constants $c_i > 0$, the PoS from Sect. 6 instantiated with the graphs from Theorem 1 is a*

$$(c_1(N/\log N), c_2(N/\log N), \infty) \text{-}\textit{secure PoS} . \tag{7}$$

Instantiated with the graphs from Theorem 2 it is a

$$(c_3 N, \infty, c_4 N)\text{-}\textit{secure PoS} . \tag{8}$$

Efficiency, measured as outlined in Sect. 3.2, is summarized in the table below where γ is the statistical security parameter

	communication	*computation* P	*computation* V
PoS Eq. (7) Initialization	$O(\gamma \log^2 N)$	$4N$	$O(\gamma \log^2 N)$
PoS Eq. (7) Execution	$O(\gamma \log N)$	*0*	$O(\gamma \log N)$
PoS Eq. (8) Initialization	$O(\gamma \log N \log\log N)$	$O(N \log\log N)$	$O(\gamma \log N \log\log N)$
PoS Eq. (8) Execution	$O(\gamma \log N)$	*0*	$O(\gamma \log N)$

Eq. (8) means that a successful cheating prover must either store a file of size $\Omega(N)$ (in L bit blocks) after initialization, or make $\Omega(N)$ invocations to the RO. Eq. (7) gives a weaker $\Omega(N/\log N)$ bound, but forces a potential adversary not storing that much after initialization, to use at least $\Omega(N/\log N)$ storage during the execution phase, no matter how much time he is willing to invest. This PoS could be interesting in contexts where one wants to be sure that one talks with a prover who has access to significant memory during execution.

Below we explain how security and efficiency claims in the theorem were derived. We start by analyzing the basic (inefficient verifier) PoS $(\mathsf{P}_0, \mathsf{V}_0)[G, \Lambda]$ from Figure 1 if instantiated with the graphs from Theorem 1 and 2.

Proposition 1. *For some constants $c_i > 0$, if G_N has pebbling complexity $\Omega(f(N), 0)$ according to Definition 1, then the basic PoS $(\mathsf{P}_0, \mathsf{V}_0)[G_N, \Lambda_N]$ as illustrated in Figure 1, where the distribution Λ_N samples $\Theta(\gamma)$ (for a statistical security parameter γ) vertices according to the distribution λ_N from Def. 1, is*

$$(S_0, c_1 f(N), \infty)\text{-}\textit{secure (for any } S_0 \le c_1 f(N)\textit{)} \tag{9}$$

If G_N has pebbling complexity $(0, g(N, S_0, S_1))$, then for any S_0, S_1 the PoS $(\mathsf{P}_0, \mathsf{V}_0)[G_N, \Lambda_N]$ is

$$(S_0, S_1, c_2 g(N, S_0, S_1))\text{-}\textit{secure.} \tag{10}$$

Above, secure means secure in the model from Sect. 3.1.

(The proof of appears in Appendix D.) Instantiating the above proposition with the graphs G_N from Theorem 1 and 2, we can conclude that the simple (inefficient verifier) PoS $(\mathsf{P}_0, \mathsf{V}_0)[G_N, \Lambda_N]$ is

$$(c_1 N/\log N, c_2 N/\log N, \infty) \quad \text{and} \quad (S_0, S_1, c_3 \cdot [S_0 \le \tau N] \cdot \max\{N, N^2/S_1\}) \tag{11}$$

secure, respectively (for constants $c_i > 0$, $0 < \tau < 1$ and $[S_0 < \tau N] = 1$ if $S_0 \leq \tau N$ and 0 otherwise). If we set $S_0 = \lfloor \tau N \rfloor = c_4 N$, the right side of Eq. (11) becomes $(c_4 N, S_1, c_3 \cdot \max\{N, N^2/S_1\})$ and further setting $S_1 = \infty$ $(c_4 N, \infty, c_3 N)$ As explained in Sect. 5, we can make the verifier V_0 efficient during initialization, by giving up on εN in the storage bound. We can choose ε ourselves, but must check $\Theta(\gamma/\varepsilon)$ values for consistency during initialization (for a statistical security parameter γ). For our first PoS, we set $\varepsilon = \frac{c_1}{2\log N}$ and get with $c_5 = c_1/2$ using $c_2 \geq c_1$

$$(\underbrace{c_1 \cdot N/\log N - \varepsilon \cdot N}_{=c_5 N/\log N}, \underbrace{c_2 \cdot N/\log N - \varepsilon \cdot N}_{\geq c_5 N/\log N}, \infty)$$

security as claimed in Eq. (7). For the second PoS, we set $\varepsilon = \frac{c_4}{2}$ which gives with $c_6 = c_4/2$

$$(\underbrace{c_4 N - \varepsilon N}_{\geq c_6 N}, \infty - \varepsilon N, c_3 N)$$

security, as claimed in Eq. (8). Also, note that the PoS described above are PoS as defined in Sect. 6 if instantiated with the graphs from Theorem 1 and 2, respectively.

Efficiency of the PoS Eq. (7). We analyze the efficiency of our PoS, measuring time and storage complexity as outlined in Sect. 3.2. Consider the $(c_1 N/\log N, c_2 N/\log N, \infty)$-secure construction from Eq. (7). In the first phase of the initialization, P needs roughly $4N = \Theta(N)$ computation: using that the underlying graph has max in-degree 2, computing $w(V)$ according to Eq. (2) requires N hashes on inputs of length at most $2L + \log N \leq 3L$, and P makes an additional $N - 1$ hashes on inputs of length $2L$ to compute the hash-tree. The communication and V's computation in the first phase of initialization is $\Theta(1)$ (as V just receives the root $\phi \in \{0,1\}^L$).

During the 2nd phase of the initialization, V will challenge P on α (to be determined) vertices to make sure that with probability $1 - 2^{-\Theta(\gamma)}$, at most an $\varepsilon = \Theta(1/\log N)$ fraction of the $\hat{x}_i$ are inconsistent. As discussed above, for this we have to set $\alpha = \Theta(\gamma \log N)$. Because this PoS is based on a graph with degree 2 (cf. Theorem 1), to check consistency of a $\hat{x}_i$ one just has to open 3 values. Opening the values requires to send $\log N$ values (and the verifier to compute that many hashes). This adds up to an $O(\gamma \log^2 N)$ communication complexity during initialization, V's computation is of the same order.

During execution, P opens ϕ on $\Theta(\gamma)$ positions, which requires $\Theta(\gamma \log N)$ communication (in L bit blocks), and $\Theta(\gamma \log N)$ computation by V.

Efficiency of the PoS Eq. (8). Analyzing the efficiency of the second PoS is analogous to the first. The main difference is that now the underlying graph has larger degree $O(\log\log N)$ (cf. Thm. 2), and we only need to set $\varepsilon = \Theta(1)$.

References

1. Martín Abadi, Michael Burrows, and Ted Wobber. Moderately hard and memory-bound functions. In *NDSS 2003*. The Internet Society, February 2003.
2. N. Alon and M. Capalbo. Smaller explicit superconcentrators. *Internet Mathematics*, 1(2):151–163, 2003.
3. N. Alon and F. Chung. Explicit construction of linear sized tolerant networks. *Discrete Math.*, 72:15–19, 1988.

4. Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In *Symposium on Theory of Computing, STOC 2015*, 2015.
5. Nate Anderson. Mining Bitcoins takes power, but is it an "environmental disaster"?, April 2013. `http://tinyurl.com/cdh95at`.
6. R. Arratia and L. Gordon. Tutorial on large deviations for the binomial distribution. *Bulletin of Mathematical Biology*, 51(1):125–131, 1989.
7. Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 538–557, 2014.
8. Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Provable data possession at untrusted stores. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 598–609. ACM Press, October 2007.
9. Adam Back. Hashcash. popular proof-of-work system., 1997. `http://bitcoin.org/bitcoin.pdf`.
10. Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008.
11. L. A. Bassalygo. Asymptotically optimal switching circuits. *Problems of Information Transmission*, 17(3):206–211, 1981.
12. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.
13. Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: theory and implementation. In *CCSW*, pages 43–54, 2009.
14. Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 857–866, 2014.
15. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.
16. Ran Canetti, Shai Halevi, and Michael Steiner. Mitigating dictionary attacks on password-protected local storage. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 160–179. Springer, August 2006.
17. R. Di Pietro, L.V. Mancini, Yee Wei Law, S. Etalle, and P. Havinga. Lkhw: a directed diffusion-based secure multicast scheme for wireless sensor networks. In *Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on*, pages 397–406, 2003.
18. John R. Douceur. The sybil attack. In *IPTPS*, pages 251–260, 2002.
19. Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 426–444. Springer, August 2003.
20. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, August 1993.
21. Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 37–54. Springer, August 2005.
22. Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. Key-evolution schemes resilient to space-bounded leakage. In *CRYPTO 2011*, LNCS, pages 335–353. Springer, August 2011.
23. Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In *TCC 2011*, LNCS, pages 125–143. Springer, 2011.
24. Paul Erdös, Ronald L. Graham, and Endre Szemerédi. On sparse graphs with dense long paths. Technical Report STAN-CS-75-504, Stanford University, Computer Science Dept., 1975.
25. Matthew K. Franklin and Dahlia Malkhi. Auditable metering with lightweight security. In Rafael Hirschfeld, editor, *FC'97*, volume 1318 of *LNCS*, pages 151–160. Springer, February 1997.
26. Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003.
27. Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In Matt Blaze, editor, *FC 2002*, volume 2357 of *LNCS*, pages 120–135. Springer, March 2002.
28. Vanessa Gratzer and David Naccache. Alien vs. quine. *IEEE Security & Privacy*, 5(2):26–31, 2007.
29. Iftach Haitner, Jonathan J. Hoch, Omer Reingold, and Gil Segev. Finding collisions in interactive protocols - a tight lower bound on the round complexity of statistically-hiding commitments. In *48th FOCS*, pages 669–679. IEEE Computer Society Press, October 2007.
30. Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
31. John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM*, 24(2):332–337, 1977.
32. Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In Bart Preneel, editor, *Communications and Multimedia Security*, volume 152 of *IFIP Conference Proceedings*, pages 258–272. Kluwer, 1999.

33. Ari Juels and John G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS'99*. The Internet Society, February 1999.
34. Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 584–597. ACM Press, October 2007.
35. Nikolaos P. Karvelas and Aggelos Kiayias. Efficient proofs of secure erasure. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 520–537, 2014.
36. Vladimir Kolmogorov and Michal Rolínek. Superconcentrators of density 25.3. arXiv Report 1405.7828, 2014. `http://arxiv.org/abs/1405.7828`.
37. 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.
38. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, February 2004.
39. R.C. Merkle. Method of providing digital signatures, January 5 1982. US Patent 4,309,569.
40. Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.
41. Silvio Micali and Ronald L. Rivest. Micropayments revisited. In Bart Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 149–163. Springer, February 2002.
42. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. `http://bitcoin.org/bitcoin.pdf`.
43. Sunoo Park, Krzysztof Pietrzak, Joël Alwen, Georg Fuchsbauer, and Peter Gazi. Spacecoin: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528, 2015. `http://eprint.iacr.org/528`.
44. Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. *Mathematical systems theory*, 10(1):239–251, 1976–1977.
45. Daniele Perito and Gene Tsudik. Secure code update for embedded devices via proofs of secure erasure. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS 2010*, volume 6345 of *LNCS*, pages 643–662. Springer, 2010.
46. Ronald L. Rivest and Adi Shamir. Payword and micromint: two simple micropayment schemes. In *CryptoBytes*, pages 69–87, 1996.
47. Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06*, volume 4341 of *LNCS*, pages 211–228. Springer, September 2006.
48. John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.
49. Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 334–345. Springer, May / June 1998.
50. E. Upfal. Tolerating a linear number of faults in networks of bounded degree. In *Proc. 11th PODC*, pages 83–89, 1992.
51. S. P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.
52. L. G. Valiant. Graph-theoretic properties in computational complexity. *Journal of Computer and System Sciences*, 13(3):278–285, 1976.
53. Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 294–311. Springer, May 2003.
54. Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New client puzzle outsourcing techniques for dos resistance. In *Proceedings of the 11th ACM conference on Computer and communications security*, CCS '04, pages 246–256, New York, NY, USA, 2004. ACM.

A Time-Memory Tradeoffs

In this section we will explain why the probably most simple and intuitive construction of a PoS, where the prover simply stores a table of outputs of a random function, only achieves very weak security due to time-memory tradeoffs for inverting random functions [31]. Consider the following simple PoS (P, V):

Initialization: P computes and stores a list L of tuples $(\mathcal{H}(i), i)$ for $i \in [N] = \{1, \ldots, N\}$, sorted by the first item.

Execution: During execution

- V picks a random value $i \leftarrow [N]$ and sends the challenge $c = \mathcal{H}(i)$ to $\mathsf{P}(L)$.

– On input c, $\mathsf{P}(L)$ checks if some tuple (c, j) is in L (which can be done in $\log(N)$ time as L is sorted), and sends back j in this case.
– V accepts if $j = i$.

Intuitively, in order to make the verifier accept with constant probability, a cheating prover $\tilde{\mathsf{P}}$ must either store a large fraction of the outputs $\mathcal{H}(i)$ (and thus use $N_0 = \Theta(N)$ storage), or search for the i satisfying $\mathcal{H}(i) = c$ by brute force (and thus use $T = \Theta(N)$ time). One thus might be tempted to conjecture that this is indeed an $(c_1 N, \infty, c_2 N)$-secure PoS for some constants $c_1, c_2 > 0$.

Unfortunately, this is not the case due to existing time-memory trade-offs for inverting random functions. Hellman [31] showed that one can invert any function $\mathcal{X} \to \mathcal{Y}$ with input domain of size $|\mathcal{X}| = N$ with constant probability in time $\Theta(N^{2/3})$ using $\Theta(N^{2/3})$ storage. This means that this PoS is not even $(c_1 N^{2/3}, \infty, c_2 N^{2/3})$-secure[16] for some $c_1, c_2 > 0$. Moreover, the $\Theta(N^{2/3})$ storage required to break the function in $\Theta(N^{2/3})$ time can be initialized in time linear in N with small hidden constants, thus this attack is very much practical.

B Online-Polls.

As another illustrative example for an (interactive, single-prover) PoS, consider on-line polling systems, like the one that is used by Wikipedia to reach consensus amongst the editors of an article. Currently, such systems do not offer sufficient protection against malicious users creating many fake identities in order to vote multiple times. A natural remedy to this problem is to link voting with a proof of work. This however is problematic, since honest users would typically not be willing to devote significant amount of their processor times to such a vote, whereas a party having a strong interest in obstructing the vote might well do so.

To give an numerical example, if the PoW requires 5 min to compute, then a dishonest player can cast almost 2000 "fake" votes in a week. If one uses PoS instead, then the situation (while still not perfect) is much better, as to achieve a similar result an adversary would need to buy a significant amount of storage. For example, if we require a user to dedicate 100 GB of disk-space in order to participate in votes, then in order to be able to cast 2000 votes, an adversary would need to invest in 200 TB of disk space, which may be prohibitive for a typical internet vandal.

C Proof of Theorem 2

For a graph $G = (V, E)$ and a subset $X \subseteq V$ we denote

– $G[X]$ to be the subgraph of G induced by X;
– $G \setminus X = G[V - X]$ to be the graph obtained by removing vertices in X;
– $\Pi_G(X)$ to be the set of ancestors of nodes in X, i.e. the set of vertices $v \in V$ from which set X is reachable in G.

[16] Recall that setting $N_1 = \infty$ just means we put no bound no the storage used during execution, but of course N_1 is upper bounded by the running time during execution plus the initial storage, so we could replace ∞ with $c_3 N^{2/3}$ (for some $c_3 > 0$) here without affecting the statement.

Some of the graphs considered in this section will implicitly come with the sets of *inputs* and *outputs*, which will be denoted as $V^+ \subseteq V$ and $V^- \subseteq V$ respectively. In such cases subgraph $G[X]$ will have inputs $V^+ \cap X$ and outputs $V^- \cap X$ (and similarly for $G \setminus X$). We also denote $\Pi^+_G(X) = \Pi_G(X) \cap V^+$ to be the set of input vertices from which set X is reachable in G. If $X = \{v\}$ then we write the sets as $\Pi_G(v)$ and $\Pi^+_G(v)$. We generally use the following convention: a subscript without parentheses denotes the number of nodes of a directed acyclic graph (e.g. G_N), while a subscript in parentheses denotes the number of inputs and outputs (e.g. graphs $R^d_{(m)}$ and $C_{(m)}$ below).

C.1 Building blocks

Our construction will rely on three building blocks. The first one is a bipartite random graph $R^d_{(m)} \in \mathcal{G}(2m, d)$ with $|V^+| = m$ inputs and $|V^-| = m$ outputs generated as follows: for each output vertex $v \in V^-$ select vertices $u_1, \ldots, u_d \in V^+$ uniformly at random and add edges $(u_1, v), \ldots, (u_d, v)$ to $R^d_{(m)}$. Note, we allow repetitions among $u_1, \ldots, u_d$. Graph $R^d_{(m)}$ is known to be a good *expander* graph with a high probability [52]; we refer to Section C.3 for a definition of expanders.

Our next building block is *superconcentrator* graphs.

Definition 2. *A directed acyclic graph $C_{(m)} = (V, E)$ with inputs V^+ and outputs V^- of size $|V^+| = |V^-| = m$ is called a* superconcentrator *if for every $k \in [m]$ and every pair of subsets $A \subseteq V^+$, $B \subseteq V^-$ of size $|A| = |B| = k$ there exist k vertex disjoint paths in $C_{(m)}$ from A to B.*

A family of superconcentrators $C_{(m)}$ is called *linear* if it has $\Theta(m)$ nodes and edges and its maximum in-degree is bounded by a constant. The existence of such superconcentrators was first shown by Valiant [53]; they used $(270 + o(1))m$ edges. The constant was successively improved in a long series of works. Currently, the smallest construction is claimed by Kolmogorov and Rolínek [**?**]; it uses $(25.3 + o(1))m$ edges, and relies on a probabilistic argument. There are also explicit constructions of linear superconcentrators, e.g. by Alon and Capalbo [2] with $(44+o(1))m$ edges.

Our third tool is graphs of Erdös, Graham and Szemerédi [24] with *dense long paths*.

Theorem 4 ([24]). *There exists a family of directed acyclic graphs $D_t = ([t], E_t)$ with t vertices and $\Theta(t \log t)$ edges (of the form (i, j) with $i < j$) that satisfy the following for some constant $\eta \in (0, 1)$ and a sufficiently large t:*

- *For any subset $X \subseteq [t]$ of size at most ηt graph $D_t \setminus X$ contains a path of length at least ηt.*

Note that the construction in [24] is randomized. In this paper we use this graph for $t = O(\log N)$, therefore the property above can be checked explicitly in time polynomial in N. This gives a Las Vegas algorithm for constructing graphs D_t with a polynomial expected running time[17].

We can also show the following.

[17] More precisely, the construction in [24] uses graphs with certain properties (see their first lemma, conditions (i)-(iii)). They show that certain random graphs satisfy (i)-(iii) with probability $\Theta(1)$. Properties (i)-(iii) can be checked in time $2^{\Theta(t)}$ which is polynomial in N. We can thus first compute graphs satisfying (i)-(iii) with a Las Vegas algorithm, and then use them to build D_t.

Proposition 2. *Family D_t in Theorem 4 can be chosen so that the maximum in-degree is $\Theta(\log t)$.*

Proof. Consider graph $D_t = (V, E)$ from Theorem 4, with $V = [t]$ and $|E| \le ct \log t$. For a node $v \in V$ let T_v be the subgraph of D_t containing node v, its predecessors $\pi(v)$ and all edges from $\pi(v)$ to v. Note, D_t is an edge-disjoint union of graphs T_v over $v \in V$.

Let d^* be the smallest even integer satisfying $2^{d^*} \ge t$. Transform tree T_v to a tree T'_v as follows: if $d_v = |\pi(v)| \le d^*$ then $T'_v = T_v$, otherwise make T'_v a tree with the root v and the leaves $\pi(v)$ such that the degree of all non-leaf nodes belongs to $[d^*/2, d^*]$. (Such tree can be obtained from T_v by repeatedly "splitting" nodes with more than d^* children.) Nodes of T_v will be called "old" and other nodes of T'_v will be called "new"; the new nodes are unique to T'_v. Let D'_t be the union of graphs T'_v over $v \in V$.

Let n_v be the number of new nodes in T'_v. If $d_v \le d^*$ then $n_v = 0$, otherwise

$$n_v \le \sum_{i=1}^{\infty} \frac{d_v}{(d^*/2)^i} = \alpha d_v \ , \qquad \alpha = \frac{2}{d^*(1 - 2/d^*)} = \frac{2}{\log t + o(\log t)}$$

The total number of new nodes is

$$n = \sum_{v \in V} n_v \le \alpha \sum_{v \in V} d_v = \alpha |E| \le \alpha ct \log t = (2c + o(1))t$$

Therefore, graph D'_t has $\Theta(t)$ nodes and maximum in-degree $d^* = \Theta(\log t)$.

Let us show that for any subset $X' \subseteq V'$ with $|X'| \le \eta t$ graph D'_t contains a path of length at least ηt that does not intersect X'; this will imply the main claim of the proposition. Define set $X \subseteq V$ via $X = \{\phi(v) \mid v \in X\}$ where mapping $\phi : V' \to V$ is the identity for old nodes, and maps new nodes in T'_v to v. Clearly, $|X| \le \eta t$. By Theorem 4, graph D_t contains a path P of length at least ηt. This path can be naturally mapped to path P' in D'_t (P' passes through the vertices of P and possibly through some new nodes). It can be seen that P' does not intersect X' and the length of P' is the same or larger than the length of P.

□

C.2 Construction of G_N

We are now ready to present our construction for Theorem 2. For integers $m, t, d > 0$ define graph $G^d_{(m,t)}$ as follows:

- Add $2mt$ nodes $V_0 \cup V_1 \cup \ldots \cup V_t$ where $|V_0| = mt$ and $|V_1| = \ldots = |V_t| = m$. Denote $\tilde{V} = V_1 \cup \ldots \cup V_t$ (this will be the set of challenges).
- Add a copy of superconcentrator $C_{(mt)}$ from V_0 to $\tilde{V}$, i.e. identify the inputs of $C_{(mt)}$ with nodes in V_0 (using an arbitrary permutation) and the outputs of $C_{(mt)}$ with nodes in $\tilde{V}$ (again, using an arbitrary permutation).
- For each edge (i, j) of graph D_t add a copy of graph $R^d_{(m)}$ from V_i to V_j.

It can be seen that $G^d_{(m,t)} \in \mathcal{G}(2mt, O(d\log t))$ (we assume that graph D_t has been chosen as in Proposition 2).

Graph $G_N = (V,E)$ for a sufficiently large N will be defined as $G^d_{(m,t)}$ for certain values m,t,d (plus "dummy" vertices to make the number of nodes in G_N to equal N). We set $t = \lfloor \mu \log N \rfloor$ and $m = \lfloor N/(2t) \rfloor$ where $\mu > 0$ is a certain constant. The family of graphs G_N is now completely defined (except for the value of constants μ, d). Note that $G_N \in \mathcal{G}(N, O(\log\log N))$ since d is constant.

Remark 2 There are certain similarities between our construction and the construction of Dwork, Naor and Wee [21]. They connect bipartite expander graphs consecutively, i.e. instead of graph D_t they use a chain graph with t nodes. Set V_0 in their construction has size m, and instead of $C_{(mt)}$ an extra graph is added from V_0 to V_1 (which is either a stack of superconcentrators or a graph from [44]). Dwork et al. give an intuitive argument that removing at most τm nodes from each layer $V_1, \ldots, V_t$ (for some constant $\tau < 1$) always leaves a graph which is "well-connected": informally speaking, many nodes of V_1 are still connected to many nodes of V_t. However, this does not hold if more than $m = \Theta(N/\log N)$ nodes are allowed to be removed: by placing initial pebbles on, say, the middle layer $V_{t/2}$ player 1 can completely disconnect V_1 from V_t.

In contrast, in our construction removing any $\tau' N$ nodes still leaves a graph which is "well-connected". Our argument is as follows. If constant τ' is sufficiently small then there can be at most ηt layers with more than τm initial pebbles (for a given constant $\tau < 1$). By the property of D_t, there exists a sufficiently long path P in D_t that avoids those layers. We can thus use the argument above for the subgraph corresponding to P. We split P into three parts of equal size, and show that many nodes in the first part are connected to many nodes in the third part.

Remark 3 As an alternative, we could have omitted superconcentrator $C^{(mt)}$ in the construction above. As will become clear from the next two sections, the resulting family of graphs would have a pebbling complexity

$$\Omega(0, [S_0 < \tau N] \cdot N) \tag{12}$$

for some constant $\tau \in (0,1)$ (with probability at least $1 - 2^{-\Theta(N/\log N)}$, for appropriate values of d, μ). However, we currently do not have any bounds on the number of pebbles S_1 for such graphs; the purpose of adding $C_{(mt)}$ was to get such bounds.

Graphs without $C_{(mt)}$ could be used if the amount of additional storage in the execution stage does not matter for a particular application.

C.3 Robust expanders

In order to analyze the construction above, we define the notion of a *robust expander*.

Definition 3. *Consider graph $G = (V,E)$ with inputs V^+ and outputs V^-, values $a,b,c > 0$ and an interval $K = [k_{\min}, k_{\max}]$.*
*(a) G is called a (K,c)-*expander *if for every set of outputs $X \subseteq V^-$ of size $|X| \in K$ there holds $|\Pi^+_G(X)| \geq c|X|$.*
(b) G is called a robust (a,b,K,c)-expander *if for every set of non-output vertices $A \subseteq V - V^-$*

of size $|A| \leq a$ there exists a set of outputs $B \subseteq V^-$ of size $|B| \leq b$ such that graph $G \setminus (A \cup B)$ is a (K, c)-expander.

By (k, c)-expanders and robust (a, b, k, c)-expanders we will mean expanders with the interval $K = [1, k]$. Intuitively, robust expansion means that the expansion property of the graph is fault-tolerant: it survives (for a large subgraph) when a constant fraction of nodes is removed.

It is known [52] that for a sufficiently large d graph $R^d_{(m)}$ is an expander (for appropriate parameters) with a high probability. We show that it is also a robust expander; a proof is given in Section C.5. [18]

Theorem 5. *There exist constants $\alpha, \kappa, \gamma \in (0, 1)$ and integer $d > 0$ with $\gamma d > 1$ such that graph $R^d_{(m)}$ is a robust $(\alpha m, \frac{1}{2}\alpha m, \kappa \frac{m}{d}, \gamma d)$-expander with probability at least $1 - 2^{-\Theta(m)}$.*

From now on we fix values $\alpha, \kappa, \gamma, d$ from Theorem 5. We can now specify value μ used in the construction of G_N: we require that

$$\mu \geq \frac{3}{\eta \log(\gamma d)} \tag{13}$$

Consider graph $G_N = (V, E)$ that was obtained from graph $G^d_{(m,t)}$. Let G be the subgraph of G_N induced by the set $\tilde{V} = V_1 \cup \ldots \cup V_t$ of size $|\tilde{V}| = mt$. From now on we assume that graph $R^d_{(m)}$ used in the construction of G_N is a robust $(\alpha m, \frac{1}{2}\alpha m, \kappa \frac{m}{d}, \gamma d)$-expander; by Theorem 5 this holds with probability at least $1 - 2^{-\Theta(m)} = 1 - 2^{-\Theta(N/\log N)}$.

Theorem 6. *For any subset $U \subseteq \tilde{V}$ of size $|U| \leq \frac{1}{2}\eta\alpha \cdot mt$ there exist at least $\frac{1}{3}\eta(1-\alpha) \cdot mt - O(m)$ vertices $v \in \tilde{V} - U$ satisfying $|\Pi_{G \setminus U}(v)| \geq \frac{1}{3}\eta\kappa\gamma \cdot mt - O(m)$.*

Proof. Let $Q \subseteq [t]$ be the set of indices i satisfying $|V_i \cap U| \geq \frac{1}{2}\alpha m$. We have

$$\tfrac{1}{2}\eta\alpha mt \geq |U| \geq |Q| \cdot \tfrac{1}{2}\alpha m \qquad \Rightarrow \qquad |Q| \leq \eta t$$

By the property of Theorem 4 graph D_t contains directed path P of length at least ηt that does not intersect Q. Thus, for each node i of P we have $|V_i \cap U| \leq \frac{1}{2}\alpha m$.

For each node i of P we define set U_i with $V_i \cap U \subseteq U_i \subseteq V_i$ and $|U_i| \leq \alpha m$ using the following recursion:

- If i is the first node of P then set $U_i = V_i \cap U$.
- Consider edge (i, j) of P for which set U_i has been defined. Denote $R_{ij} = G[V_i \cup V_j]$; it is a copy of $R^d_{(m)}$. By the robust expansion property there exists subset $B_j \subseteq V_j$ with $|B_j| \leq \frac{1}{2}\alpha m$ such that graph $R_{ij} \setminus (U_i \cup B_j)$ is a $(\kappa \frac{m}{d}, \gamma d)$-expander.
  We define $U_j = (V_j \cap U) \cup B_j$, then $|U_j| \leq |V_j \cap U| + |B_j| \leq \frac{1}{2}\alpha m + \frac{1}{2}\alpha m = \alpha m$.

[18] Note that the robust expansion property has been formulated in [21, Section 4]. Namely, they say that "*in any good enough expander if up to some constant (related to the expansion) fraction of nodes are deleted, then one can still find a smaller expander (of linear size) in the surviving graph*". To support this claim, they cite [3, 51]. However, we were unable to find such statement in these references.

We believe that inferring the robust expansion property of $R^d_{(m)}$ just from the ordinary expansion is indeed possible, but with a worse bound on the probability and with worse constants compared to what we have in Theorem 5 and its proof.

Note that graph $R_{ij} \setminus (U_i \cup U_j)$ is also a $(\kappa \frac{m}{d}, \gamma d)$-expander: it is obtained from $R_{ij} \setminus (U_i \cup B_j)$ by removing a subset of outputs, and such operation preserves the expansion property.

Let $I \subseteq [t]$ be the first $\lfloor \frac{1}{3}\eta t \rfloor$ nodes of P and $J \subseteq [t]$ be the last $\lfloor \frac{1}{3}\eta t \rfloor$ nodes. Consider vertex $v_\circ \in V_j - U_j$ for index $j_\circ \in J$. We will show $|\Pi_{G \setminus U}(v_\circ)| \geq \frac{1}{3}\eta\kappa\gamma \cdot mt - O(m)$. This will imply the theorem since the number of such vertices is at least $|J| \cdot (m - \alpha m) \geq \frac{1}{3}\eta t \cdot (1-\alpha)m - O(m)$.

For nodes i of path P denote

$$X_i = \Pi_{G \setminus U}(v_\circ) \cap (V_i - U_i) \tag{14}$$

For a node $i \leq j_\circ$ of P let $\ell(i)$ be the distance from i to $j_\circ$ along P (with $\ell(j_\circ) = 0$). We use induction on $\ell(i)$ to show that

$$|X_i| \geq \min\{(\gamma d)^{\ell(i)}, \lfloor \kappa \tfrac{m}{d} \rfloor \cdot \gamma d\} \tag{15}$$

For $\ell(i) = 0$ the claim is trivial (since $X_{j_\circ} = \{v_\circ\}$). Suppose it holds for j, and consider edge (i, j) of path P. Note that $\ell(i) = \ell(j) + 1$. By construction, graph $R_{ij} \setminus (U_i \cup U_j)$ is a $(\kappa \frac{m}{d}, \gamma d)$-expander. Furthermore, $\Pi^+_{R_{ij} \setminus (U_i \cup U_j)}(X_j) \subseteq X_i$. Together with the induction hypothesis this implies the claim of the induction step:

$$\begin{aligned}|X_i| &\geq |\Pi^+_{R_{ij} \setminus (U_i \cup U_j)}(X_j)| \geq \gamma d \cdot \min\{|X_j|, \lfloor \kappa \tfrac{m}{d} \rfloor\} \\ &\geq \gamma d \cdot \min\{(\gamma d)^{\ell(j)}, \lfloor \kappa \tfrac{m}{d} \rfloor \cdot \gamma d, \lfloor \kappa \tfrac{m}{d} \rfloor\} = \min\{(\gamma d)^{\ell(j)+1}, \lfloor \kappa \tfrac{m}{d} \rfloor \cdot \gamma d\}\end{aligned}$$

We have proved eq. (15) for all nodes i of P. Now consider node $i \in I$. We have $\ell(i) \geq \frac{1}{3}\eta t$ and also $t = \lfloor \mu \log N \rfloor \geq \mu \log m \geq \frac{3}{\eta \log(\gamma d)} \log m$. Therefore,

$$(\gamma d)^{\ell(i)} \geq (\gamma d)^{\frac{1}{3}\eta \cdot \frac{3}{\eta \log(\gamma d)} \log m} = m$$

and so the minimum in (15) is achieved by the second expression $\lfloor \kappa \frac{m}{d} \rfloor \cdot \gamma d = \kappa\gamma m - O(1)$. (Note, we must have $\kappa\gamma \leq 1$, otherwise we would get $|X_i| > m$ - a contradiction). We obtain the desired claim:

$$|\Pi_{G \setminus U}(v_\circ)| \geq \textstyle\sum_{i \in I} |X_i| \geq (\frac{1}{3}\eta t - O(1)) \cdot (\kappa\gamma m - O(1)) = \frac{1}{3}\eta\kappa\gamma \cdot mt - O(m)$$

□

C.4 Proof of Theorem 2: a wrap-up

Using Theorem 6 and a result from [37], we can now show that graphs G_N have a pebbling complexity $\Omega(0, [S_0 < \tau N] \cdot \max\{N, N^2/S_1\})$ where $\tau = \frac{1}{2}\eta\alpha \cdot \min_N \frac{mt}{N} \in (0, 1)$. We define λ_N as the uniform probability distribution over vertices $c \in \tilde{V}$.

Assume that the set initial pebbles $U \subseteq V$ chosen by player 1 has size $|U| = S_0 < \tau N$, otherwise the claim is trivial. Fix a constant $\rho \in (0, \frac{1}{3}\eta\kappa\gamma)$. We say that a sample $c \leftarrow \lambda_N$ is *good* if $|\Pi_{G \setminus U}(c)| > \rho \cdot mt$. Theorem 6 implies that c is good with probability at least δ for some constant $\delta > 0$ (assuming that N is sufficiently large).

Let us assume that c is good. To pebble c, one must pebble at least $\rho \cdot mt$ nodes of $\tilde{V}$; this requires time $T = \Omega(N)$. Next, we use the following standard result about superconcentrators; it is a special case of Lemma 3.2.1 in [37].

Lemma 2. *In order to pebble $2S_1 + 1$ outputs of superconcentrator $C_{(mt)}$, starting and finishing with a configuration of at most S_1 pebbles, at least $mt - 2S_1$ different inputs of the graph have to be pebbled and unpebbled.*

By applying this lemma $\lfloor \rho \cdot mt/(2S_1 + 1) \rfloor$ times we conclude that pebbling c with at most S_1 pebbles requires time $T = \Omega(N^2/S_1)$. The theorem is proved.

C.5 Proof of Theorem 5 (robust expansion of $R^d_{(m)}$)

In this section we denote graph $R^d_{(m)}$ as $G = (V, E)$ (with inputs V^+ and outputs V^-). For a set of output nodes $X \subseteq V^-$ let $\pi(X) = \Pi^+_G(X) = \bigcup_{v \in X} \pi(v)$ be the set of predecessors of nodes in X.

We will use the following fact about the binomial distribution (see [6]).

Theorem 7. *Suppose that $X_1, \ldots, X_N$ are independent $\{0, 1\}$-valued variables with $p = p(X_i = 1) \in (0, 1)$. If $p < \frac{M}{N} < 1$ then*

$$F_{\geq}(M; N, p) \stackrel{\text{def}}{=} Pr[\, \textstyle\sum_{i=1}^{N} X_i \geq M \,] \leq \exp\left(-N \cdot H\left(\frac{M}{N}, p\right)\right)$$

where

$$H(a, p) = a \ln \frac{a}{p} + (1 - a) \ln \frac{1 - a}{1 - p}$$

Let us fix values $\alpha, \beta, \kappa, \gamma, \delta \in (0, 1)$ and integer $d > 0$ satisfying

$$\beta < \tfrac{1}{2}\alpha \tag{16a}$$
$$\alpha < \delta - \gamma \tag{16b}$$
$$q < \beta \qquad \text{where} \quad q = \exp(-d \cdot H(\delta - \gamma, \alpha)) \tag{16c}$$
$$H(\beta, q) - \ln \tfrac{e}{\alpha} > 0 \tag{16d}$$
$$\kappa < \bar{\delta} \qquad \text{where} \quad \bar{\delta} = 1 - \delta \tag{16e}$$
$$\bar{\delta} d > 1 \tag{16f}$$
$$- \ln(de) + d \cdot [\bar{\delta} \ln \bar{\delta} + \delta \ln \delta] + (\bar{\delta} d - 1) \cdot \ln \tfrac{1}{\kappa} > 0 \tag{16g}$$

Examples of feasible parameters are given in Table 1. We will show that Theorem 5 holds for any values satisfying (16). [19]

Throughout this section we denote

$$k_{\max} = \kappa \tfrac{m}{d} \tag{17a}$$
$$k_{\min} = \lfloor \min\{(\tfrac{1}{2}\alpha - \beta)m, \tfrac{1}{2}k_{\max}\} \rfloor \tag{17b}$$
$$K = [k_{\min}, k_{\max}] \tag{17c}$$

We also introduce the following definition.

[19] We conjecture that the constants could be improved if instead of $R^d_{(m)}$ we used a random bipartite graph (with multi-edges allowed) in which degrees of nodes in both V^+ in V^- equal d (i.e. a union of d random permutation graphs). Expansion properties of such graphs were analyzed in [11].

α	κ	γ	δ	d
0.12	$1/8$	$1/4$	0.676	≥ 190
0.19	$1/16$	$1/8$	0.757	≥ 67
0.22	$1/32$	$1/16$	0.800	≥ 43
0.24	$1/64$	$1/32$	0.827	≥ 34

Table 1. Feasible parameters satisfying (16). We always use $\beta = \frac{1}{2}\alpha - \epsilon$ for a sufficiently small $\epsilon > 0$.

Definition 4. *For a set of inputs $A \subseteq V^+$ define set of outputs B_A via*

$$B_A = \{v \in V^- \mid |\pi(v) \cap A| \geq (\delta - \gamma)d\} \tag{18}$$

Graph G is a backward (a, b)-expander *if $|B_A| \leq b$ for any set $A \subseteq V^+$ of size $|A| \leq a$.*

Theorem 5 will follow from the following three facts.

Lemma 3. *Suppose that G is a backward $(\alpha m, \beta m)$-expander and also a $(K, \delta d)$-expander.*
(a) It is a robust $(\alpha m, \beta m, K, \gamma d)$-expander.
(b) It is a robust $(\alpha m, \beta m + k_{\min}, k_{\max}, \gamma d)$-expander.

Lemma 4. *There exists constant $c_1 > 0$ such that*

$$Pr[\text{ } G \text{ is not a backward } (\alpha m, \beta m)\text{-expander }] \leq 2^{-c_1 m}$$

Lemma 5. *There exists constant $c_2 > 0$ such that*

$$Pr[\text{ } G \text{ is not a } (K, \delta d)\text{-expander }] \leq 2 \cdot 2^{-c_2 k_{\min}}$$

As a corollary, we obtain that G is a robust $(\alpha m, \beta m + k_{\min}, \kappa \frac{m}{d}, \gamma d)$-expander with probability at least $1 - 2^{-c_1 m} - 2 \cdot 2^{-c_2 k_{\min}}$. Therefore, it is also a robust $(\alpha m, \frac{1}{2}\alpha, \kappa \frac{m}{d}, \gamma d)$-expander with this probability, since $\frac{1}{2}\alpha \geq \beta m + k_{\min}$. By observing that $c_1 m = \Theta(m)$ and $c_2 k_{\min} = \Theta(m)$ we get Theorem 5.

The remainder of this section is devoted to the proof of Lemmas 3-5.

Proof of Lemma 3(a) Given set $A \subseteq V^+$ of size $|A| \leq \alpha m$, we construct set B via $B = B_A$; the backward expansion property implies that $|B| \leq \beta m$. Let us show that graph $G \setminus (A \cup B)$ is a $(K, \gamma d)$-expander. Consider set $X \subseteq V^- - B$ with $|X| = k \in K$. We can partition $\pi(X)$ into disjoint sets $Y = \pi(X) \cap A$ and $Z = \pi(X) - A$. For each $v \in X$ denote $Y_v = \pi(v) \cap A$, then $|Y_v| \leq (\delta - \gamma)d$ (since $v \notin B_A$). The desired inequality can now be derived as follows:

$$|Z| = |Y \cup Z| - |Y| = |\pi(X)| - |\bigcup_{v \in X} Y_v| \geq \delta k d - \sum_{v \in X} |Y_v| \geq \delta k d - |X| \cdot (\delta - \gamma)d = \gamma k d$$

Proof of Lemma 3(b) Consider set $A \subseteq V^+$ of size $|A| \leq \alpha m$. By Lemma 3(a) there exists set $B \subseteq V^-$ of size $B \leq \beta m$ such that graph $G \setminus (A \cup B)$ is a $(K, \gamma d)$-expander. We will denote this graph as $\hat{G}$, and its inputs and outputs as $\hat{V}^+ = V^+ - A$ and $\hat{V}^- = V^- - B$ respectively. For a set $X \subset \hat{X}^-$ let $\hat{\pi}(X) \subseteq \hat{V}^+$ be the set of predecessors of X in $\hat{G}$.

We will show that there exists set $\hat{B} \subseteq \hat{V}^-$ of size $|\hat{B}| \leq k_{\min} - 1$ such that graph $\hat{G} \setminus \hat{B}$ is a $(k_{\max}, \gamma d)$-expander; this will imply the claim of the lemma. In fact, it suffices to show that it is a $(k_{\min} - 1, \gamma d)$-expander, since we already know that $\hat{G} \setminus \hat{B}$ is a $([k_{\min}, k_{\max}], \gamma d)$-expander for any $\hat{B} \subseteq \hat{V}^-$.

We construct set $\hat{B}$ using the following greedy algorithm:

– set $\hat{B} := \emptyset$;
– while there exists subset $X \subseteq \hat{V}^- - \hat{B}$ such that $|X| \le k_{\min} - 1$ and $|\hat{\pi}(X)| < \gamma d|X|$, update $\hat{B} := \hat{B} \cup X$.

By construction, upon termination we get set $\hat{B}$ such that graph $\hat{G} \setminus \hat{B}$ is a $(k_{\min} - 1, c)$-expander. To prove the lemma, it thus suffices to show that $|\hat{B}| \le k_{\min} - 1$. Suppose that this is not the case. Let $\hat{B}' \subseteq \hat{B}$ be the first subset during the execution of the algorithm whose size exceeds $k_{\min} - 1$, then $k_{\min} \le |\hat{B}'| \le 2k_{\min} \le k_{\max}$. We have $|\hat{B}'| \in K$, so the $(K, \gamma d)$-expansion property of $\hat{G}$ implies that $|\hat{\pi}(\hat{B}')| \ge \gamma d|\hat{B}'|$. However, by inspecting the algorithm above we conclude that $|\hat{\pi}(\hat{B}')| < \gamma d|\hat{B}'|$ - a contradiction.

Proof of Lemma 4 Denote $\alpha' = \lfloor \alpha m \rfloor / m \le \alpha$. Clearly, it suffices to prove the backward expansion property only for sets of inputs $A \subseteq V^+$ of size $|A| = \lfloor \alpha m \rfloor = \alpha' m$. Let us fix such set A. For an output vertex $v \in V^-$ denote $q' = Pr[v \in B_A]$. It is the probability that $|\pi(v) \cap A| \ge (\delta - \gamma)d$. Each node $u \in \pi(v)$ falls in A with probability $\alpha' = |A|/m$, therefore

$$q' = F_{\ge}((\delta - \gamma)d; d, \alpha') \le F_{\ge}((\delta - \gamma)d; d, \alpha) \le \exp(-d \cdot H(\delta - \gamma, \alpha)) = q$$

(We used the fact that $\delta - \gamma > \alpha$ by (16b).) From the inequalities above and from (16c) we get $q' \le q < \beta$, therefore

$$Pr[|B_A| \ge \beta m] = F_{\ge}(\beta m; m, q') \le F_{\ge}(\beta m; m, q) \le \exp\left(-m \cdot H(\beta, q)\right)$$

We now use a union bound:

$$\begin{aligned} Pr[G \text{ is not a backward } (\alpha m, \beta m)\text{-expander}] &\le \sum_{A \subseteq V^+ : |A| = \alpha' m} Pr[|B_A| \ge \beta m] \\ &\le \binom{m}{\alpha' m} \exp\left(-m \cdot H(\beta, q)\right) \le \left(\frac{me}{\alpha m}\right)^{\alpha m} \exp\left(-m \cdot H(\beta, q)\right) \\ &= \exp\left(-m \cdot (H(\beta, q) - \ln \tfrac{e}{\alpha})\right) \end{aligned}$$

Combined with condition (16d), this implies the claim.

Proof of Lemma 5 We follow the argument from [52], only with different constants. Consider integer $k \in K = [k_{\min}, \kappa \frac{m}{d}]$, and let p_k be the probability that there exists set $X \subseteq V^-$ of size exactly k with $|\pi(X)| < \delta d \cdot k$. We prove below that $p_k \le 2^{-ck}$ for some constant $c > 0$. This will imply Lemma 5 since then

$$Pr[\ G \text{ is not a } (K, \delta d)\text{-expander}\ ] \le \sum_{k=k_{\min}}^{\lfloor \kappa \frac{m}{d} \rfloor} 2^{-ck} < 2^{-ck_{\min}} \sum_{i=0}^{\infty} 2^{-i} = 2 \cdot 2^{-ck_{\min}}$$

Recall that we denoted $\bar{\delta} = 1 - \delta$. Let us also denote $\lambda = \frac{kd}{m}$. Note, condition $k \le \kappa \frac{m}{d}$ implies that $\lambda \le \kappa$.

Consider a fixed set $X \subseteq V^-$ of size k. Let us estimate the probability that $|\pi(X)| < \bar{\gamma} kd$. Set $\pi(X)$ contains a union of kd independent random variables $J_1, \ldots, J_{kd}$ where each J_i is a node in V^+ chosen uniformly at random. We can imagine these nodes $J_1, \ldots, J_{kd}$ being chosen in sequence. Call J_i a repeat if $J_i \in \{J_1, \ldots, J_{i-1}\}$. Then the probability that J_i is a repeat, even conditioned on $J_1, \ldots, J_i - 1$, is at most $\frac{i-1}{m} \le \frac{kd}{m} = \lambda$.

Let $\hat{J}_1, \ldots, \hat{J}_{kd}$ be independent random variables that take values "repeat" and "no repeat", with $Pr[\hat{J}_i = \text{repeat}] = \lambda$. Then

$$\begin{aligned} Pr[\, |\pi(X)| < \delta kd \,] &\le Pr[\text{ there are at least } \lfloor \bar{\delta} kd + 1 \rfloor \text{ repeats among } J_1, \ldots, J_{kd} \,] \\ &\le Pr[\text{ there are at least } \lfloor \bar{\delta} kd + 1 \rfloor \text{ repeats among } \hat{J}_1, \ldots, \hat{J}_{kd} \,] \\ &= F_{\ge}(\lfloor \bar{\delta} kd + 1 \rfloor; kd, \lambda) \;\le\; F_{\ge}(\bar{\delta} kd; kd, \lambda) \;\le\; \exp\left(-kd \cdot H(\bar{\delta}, \lambda)\right) \end{aligned}$$

The number of subsets $X \subseteq V^-$ of size k is $\binom{m}{k} \le \left(\frac{me}{k}\right)^k = \left(\frac{de}{\lambda}\right)^k$. Therefore,

$$\begin{aligned} p_k &\le \sum_{X \subseteq V^- : |X| = k} Pr[\, |\pi(X)| < \delta kd \,] \le \left(\tfrac{de}{\lambda}\right)^k \exp\left(-kd \cdot H(\bar{\delta}, \lambda)\right) \\ &= \exp\left(\, -k \cdot [\, -\ln \tfrac{de}{\lambda} + d \cdot \bar{\delta} \ln \tfrac{\bar{\delta}}{\lambda} + d \cdot \delta \ln \tfrac{\delta}{1-\lambda} \,] \, \right) \\ &= \exp\left(\, -k \cdot [\, \sigma + (\bar{\delta} d - 1) \cdot \ln \tfrac{1}{\lambda} + \delta d \cdot \ln \tfrac{1}{1-\lambda} \,] \, \right) \end{aligned} \tag{19}$$

where in σ we collected terms that do not depend on λ:

$$\sigma = -\ln(de) + d \cdot [\bar{\delta} \ln \bar{\delta} + \delta \ln \delta]$$

Note that $\bar{\delta} d - 1 > 0$ by (16f). Plugging inequalities $\ln \frac{1}{\lambda} \ge \ln \frac{1}{\kappa}$ and $\ln \frac{1}{1-\lambda} \ge 0$ into (19) gives

$$p_k \le \exp\left(\, -k \cdot [\, \sigma + (\bar{\delta} d - 1) \cdot \ln \tfrac{1}{\kappa} \,] \, \right)$$

The coefficient after k in the last expression is a positive constant according to (16g). The claim $p_k \le 2^{-ck}$ for a constant $c > 0$ is proved.

C.6 Superconcentrators are robust expanders

For completeness, in this section we show that superconcentrators are also robust expanders (for appropriate parameters). This suggests that in the construction of G_N one could replace bipartite random graph $R^d_{(m)}$ with superconcentrator $C_{(m)}$; the argument of Theorem 6 would still apply. More precisely, we have two options:

– When adding a copy of $C_{(m)}$ for edge $(i,j) \in D_t$, create unique copies of internal nodes of $C_{(m)}$ (i.e. those nodes that are neither inputs nor outputs; there are $\Theta(m)$ such nodes). Graph $G_{(m,t)}$ would then have $\Theta(mt \log t)$ nodes instead of $\Theta(mt)$ nodes. We could thus obtain a family of graphs G_N with a constant average degree and pebbling complexity

$$\Omega(0, [S_0 < \tau \tfrac{N}{\log \log N}] \cdot (\tfrac{N}{\log \log N})^2 / S_1) \tag{20}$$

for some constant $\tau > 0$.

- Share internal nodes of superconcentrators for edges (i, j) that are going to the same node $j \in [t]$. Graph $G_{(m,t)}$ would then have $\Theta(mt)$ nodes. This would give a family of graphs $G_N \in \mathcal{G}(N, O(\log\log N))$ with the same pebbling complexity as in Theorem 2.

We omit formal derivations of these claims (thus leaving them as conjectures); instead, we only prove the following result.

Theorem 8. *Suppose that values m, α, k, c satisfy $c(\alpha m + k) \leq (1-\alpha)m$. Then superconcentrator $G = (V, E)$ with $|V^+| = m$ inputs and $|V^-| = m$ outputs is a robust $(\alpha m, \alpha m, k, c)$-expander.*

We will need the following well-known property of a superconcentrator.

Lemma 6 ([44]). *If $A \subseteq V - V^-$, $B \subseteq V^-$ are subsets with $|A| < |B|$ then $|\Pi^+_{G\setminus A}(B)| \geq m - |A|$.*

Proof. If $|\Pi^+_{G\setminus A}(B)| < m - |A|$ then $|V^+ - \Pi^+_{G\setminus A}(B)| \geq |A| + 1$, so there must exist $|A| + 1$ vertex-disjoint paths between $V^+ - \Pi^+_{G\setminus A}(B)$ and B. At least one of them does not intersect A, and thus its source node belongs to $\Pi^+_{G\setminus A}(B)$ - a contradiction. $\square$

We now proceed with the proof of Theorem 8. Consider subset $A \subseteq V - V^-$ with $|A| \leq \alpha m$. We construct set $B \subseteq V^-$ using the following greedy algorithm:

- set $B := \emptyset$;
- while there exists subset $X \subseteq V^- - B$ such that $|X| \leq k$ and $|\Pi^+_{G\setminus A}(X)| < c|X|$, update $B := B \cup X$.

By construction, upon termination we get set B such that graph $G \setminus (A \cup B)$ is a (k, c)-expander. To prove the theorem, it thus suffices to show that $|B| \leq \alpha m$. Suppose that $|B| > \alpha m$. Let $B' \subseteq B$ be the first subset during the execution of the algorithm whose size exceeds αm, then $\alpha m < |B'| \leq \alpha m + k$. We have $|B'| > |A|$, so by Lemma 6 $|\Pi^+_{G\setminus A}(B')| \geq m - |A| \geq (1 - \alpha)m$. By inspecting the algorithm above we conclude that $|\Pi^+_{G\setminus A}(B')| < c|B'|$. We obtained that $c(\alpha m + k) \geq c|B'| > (1 - \alpha)m$ - a contradiction.

D Proof of Proposition 1

We start explaining the security as claimed in Eq. (9). G_N having pebbling complexity $(f(N), 0)$ means, that any player 1 who is allowed to put at most $c_1 f(N)$ (for some constant $c_1 > 0$) pebbles on the graph simultaneously must fail in pebbling a challenge vertex $c \leftarrow \lambda_N$ with probability at least δ for some constant $\delta > 0$ (even with no bound on the number of pebbling steps). If we now pick $\Theta(\gamma)$ challenge vertices $C \leftarrow \Lambda_N$ independently, then with probability $1 - (1 - \delta)^\gamma = 1 - 2^{-\Theta(\gamma)}$ we will hit a vertex that cannot be pebbled with $c_1 f(N)$ pebbles. As discussed in Section 4, this means the scheme is $(0, c_1 f(N), \infty)$-secure. Observing that $(0, S_1, \infty)$-security implies (S_0, S_1, ∞)-security for any $S_0 \leq S_1$ proves Eq. (9).

We will now show Eq. (10). G_N having pebbling complexity $(0, g(N, S_0, S_1))$ means that any player 1 who is allowed to put S_0 pebbles on the graph initially, and then use at most S_1 pebbles simultaneously making at most $c_2 g(N, S_0, S_1)$ pebbling steps, must fail in pebbling a challenge vertex $c \leftarrow \lambda_N$ with probability at least $\delta > 0$. As above, if we pick $\Theta(\gamma)$ challenges $C \leftarrow \Lambda_N$ independently, then with probability $1 - (1 - \delta)^\gamma = 1 - 2^{-\Theta(\gamma)}$ we will hit a vertex that cannot be pebbled with $c_2 g(N, S_0, S_1)$ steps. As discussed in Section 4, this means the scheme $(S_0, S_1, c_2 g(S_0, S_1, N))$-secure.