Proofs of Space

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

[1]University of Warsaw
[2]EPFL Lausanne
[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, using graphs with high "pebbling complexity" and Merkle hash-trees.

1 Introduction

Proofs of Work (PoW). Dwork and Naor [DN93] 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 [BR93], then the sender must compute an expected 2^t hashes until she finds such an σ.[1] A useful property of this PoW is that there's 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.

Proofs of Space (PoS). A proof of work is simply a means of showing that one invested a non-trivial amount of computational work related to some statement. This general principle also works with resources other than computation like real money in micropayment systems [MR02] or human attention in CAPTCHAs [vABHL03]. In this paper we put forward the concept of *proofs of space* (PoS), where only parties who dedicate a significant amount of disk space can efficiently generate proofs.

Our approach borrows ideas from [ABW03,DGN03,DNW05], who suggest to construct PoW using functions whose computation requires accessing memory many times.[2] The above-mentioned PoW can be seen as some kind of PoS, but they lack several properties we expect from a PoS, most importantly, the verifier needs to use the same amount of memory as the prover. This is not a problem in the original application of PoWs as here the space just needs to be larger than the cache of a potential malicious prover. When used a PoS, where the main resource dedicated is space, this is not an option, as here we want the verifier to use a tiny fraction of the space dedicated by the prover.

[1] 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 time point.

[2] This is motivated by the fact that the memories typically do not differ much in terms of the access time, which is in contrast with the computing speeds of CPU's that differ significantly. Hence, it is argued that this type of a proof of work would be more fair in the sense that having extremely fast CPUs does not help much, because the running time is to a large extend determined by the number of cache misses (which require "slow" memory accesses).

Nowadays users often have lots of free disk space (a standard laptop comes with a 500 GB hard disk). In such a case, running a PoS comes basically for free, except for a linear (in the dedicated storage) cost during an initialization phase for the prover.

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

Formalizing PoS. In this paper we only consider *interactive* PoS between a *single* prover P and a verifier V. We will discuss non-interactive multi-prover PoS, and the potential application of such a scheme to the Bitcoin digital currency system, in Appendix B.

As an illustrative application for an (interactive, single prover) PoS, consider an organization that offers a free email service, like gmail from Google (we will outline another example related to online polls in Section C.) To prevent that someone registers a huge number of fake-addresses for spamming, Google might require users to dedicate some nontrivial amount of disk space, say 100GB, for every address registered. Occasionally, Google will run a PoS to make sure the user really dedicates this space.

The most trivial solution would be for the verifier V (here, Google) to generate a random (and thus incompressible) 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) $\mathcal{F}$. A slightly better solution is to have V generate $\mathcal{F}$ using a pseudorandom function, in which case V doesn't have to store the entire $\mathcal{F}$, but just a short PRF key. 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[3] 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 later 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.

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 he either uses N_0 bits of storage in-between executions of the PoS *or* if $\tilde{\mathsf{P}}$ invests T time 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 Section 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 $N_1 \geq N_0$).

Outline and Our Contribution. In this paper we introduce the concept of a PoS, which we formally define in Section 2. In Section 3 we discuss and motivate the model in which we prove our constructions secure. In Section 4 we explain how to reduce the security of a simple PoS (with an inefficient verifier) to a graph pebbling game. In Section 5 we show how to use hash-trees to make the verifier in the PoS from Section 4 efficient. In Section 6 we define our final construction. In Section 7 we state our results on constructing graphs with high pebbling complexity, and in Section 8 we state our main Theorem 3 on the security and efficiency of our PoS from Section 6 if instantiated with the graphs from Section 7.

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

In Section 7 we consider two different (infinite families of) graphs with different (and incomparable) pebbling complexities. These graphs then also give PoS schemes with different parameters. Informally, one of our constructions requires a successful malicious prover to either dedicate $\Theta(N)$ storage (i.e., almost as much as the N stored by the honest prover) or otherwise to use $\Theta(N)$ time with every execution of the PoS. For our other construction we prove a somewhat weaker $\Omega(N/\log N)$ bound on the storage required by a malicious prover, but here a successful prover is forced to use at least $\Omega(N/\log N)$ storage during execution, no matter how much time he is wiling to spend during execution.

More Related Work Dwork and Naor [DN93] 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 [ABW03] 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 [DGN03,DNW05,WJHF04].

Originally put forward to counteract spamming, PoWs have a vast number of different applications such as metering web-site access [FFM], countering denial-of-service attacks [JB99,Bac97] and many more [JJ99]. An important application for PoWs are digital currencies, like the recent Bitcoin system [Nak09], or earlier schemes like the Micromint system of Rivest and Shamir [RS96]. 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 [Dou02].

A large body of work investigates the concept of **proofs of storage** or **proofs of retrievability** (cf. [GJM02,BJO09,ABC$^+$07,JK07,DPML$^+$03] 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 stores 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.

2 PoS Definition

We denote with $(\mathsf{out}_\mathsf{V}, \mathsf{out}_\mathsf{P}) \leftarrow \langle \mathsf{V}(\mathsf{in}_\mathsf{V}), \mathsf{P}(\mathsf{in}_\mathsf{P})\rangle(\mathsf{in})$ the execution of an interactive protocol between two parties P and V on shared input in, local inputs[4] in_P and in_V, and with local outputs out_V and out_P, respectively.

A proof of space (PoS) is given by a pair of interactive random access machines,[5] a prover P and a verifier V. These parties can run two kinds of protocols, a PoS initialization and a PoS execution, as defined below. The protocols are done 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 output is

$$(\Phi, S) \leftarrow \langle \mathsf{V}, \mathsf{P}\rangle(\mathsf{prm}) \quad \text{where } \Phi \text{ is short and } S \text{ is of size } N$$

V can output the special symbol $\Phi = \perp$, which means that it aborts (this will be only the case if V catches P cheating).

[4] 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.

[5] 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 takes constant time.

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.

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

As for any proof system, we require completeness and security.

Completeness: We will always require perfect completeness, i.e., for any honest prover P:

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

Security: An (N_0, N_1, T)-prover $\tilde{\mathsf{P}}$ is a prover where

N_0: The output S of $\tilde{\mathsf{P}}$ after initialization has size at most N_0.

N_1, T: The running time of $\tilde{\mathsf{P}}$ during execution is at most T. Moreover, $\tilde{\mathsf{P}}$ uses at most N_1 storage during execution. Note that we put no time or memory bound on P during initialization, N_0 only denotes the size of the final output.

A PoS (P, V) is (N_0, N_1, T)-secure if every (N_0, N_1, T)-adversary $\tilde{\mathsf{P}}$ will fail in making V accept except with some exponentially small probability[6] in a statistical security parameter γ.[7]

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

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. P must be efficient during execution, but can run in time poly(N) during initialization.[8]

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, t) prover, where n, t denote the storage and time requirements of P during execution, then *no* PoS can be

1. (N, n, t)-secure, as the honest prover "breaks" the scheme by definition.
2. $(c, I + t + c, I + t)$-secure, where c is the number of bits send 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 send 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 [BR93], 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 [CGH98,GK03,MRH04], 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 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.

[6] Below 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. The protocols in this paper are of this form.

[7] Our construction is based on a hash-function $\mathcal{H}$, which we require to be collision resistant. As assuming collision resistance for a fixed function is not meaningful [Rog06], 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.

[8] 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 3.

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[9] $\mathcal{A} = (a_1, \ldots, a_q)$ 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 is committed to a subset of the queries $\mathcal{B} \subseteq \mathcal{A}$ of size N.
2. In a second phase, $\tilde{\mathsf{P}}(S)$ is asked to output $\mathcal{H}(b)$ for some random $b \in \mathcal{B}$. $\tilde{\mathsf{P}}(S)$ 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.[10] Similarly, the storage bound $N_1 L$ during execution implies that $\mathcal{H}$ 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.

3.2 Storage and Time Complexity.

Time Complexity. Throughout, we will equate the *running time* of honest and adversarial parties by 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 Section 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 $[N] = \{1, \ldots, N\}$. With every vertex $v \in V$ we associate a value $w(v) \in \{0,1\}^L$, and for a subset $V' = \{v_1, \ldots, v_n\} \subseteq V$ 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 RO to the index v and the values of its predecessors

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

[9] 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.

[10] 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.

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 $G = (V, E)$. Note that this PoS does not yet satisfy the efficiency requirement from Section 2. In our model as discussed in Section 3.1, the only way a malicious prover $\tilde{\mathsf{P}}_0(S)$ can determine

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 Section 6).
Initialization $(S, \emptyset) \leftarrow \langle \mathsf{P}_0, \mathsf{V}_0 \rangle(\mathsf{prm})$ where $S = w(V)$.
Execution $(\mathsf{accept}/\mathsf{reject}, \emptyset) \leftarrow \langle \mathsf{V}(\emptyset), \mathsf{P}(S) \rangle(\mathsf{prm})$

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).

$w(v)$ is if $w(v) \in S_{\mathcal{H}}$ is in the encoded set of size $\leq N_0$, 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 $i \in \pi(v)$, $\tilde{\mathsf{P}}_0(S)$ will have to make even more queries recursively to learn $w(v)$.

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 $C \leftarrow \Lambda, |C| = \alpha$ 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 Hash Trees

The PoS described in the previous section does not yet meet our Definition from Section 2 as V_0 is not efficient. In this section we describe how to make the verifier efficient, using hash-trees.

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 $\mathcal{H}$.

5.1 Using hash trees to make the verifier efficient

We add an initialization phase 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 verifier 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 an option as we want V's complexity to be only polylogarithmic in N.
- Let P prove, using a proof system like computationally sound (CS) proofs [Mic00] or universal arguments [BG08], that ϕ was computed correctly. Although this 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.

5.2 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 $\tilde{x}_1, \ldots, \tilde{x}_N$.[11] 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.

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

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.

In our idealized model as discussed in Section 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 Section 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 PoS 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),[12] 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 Section 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 $C \leftarrow \Lambda, |C| = \alpha$ 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 $c \in C$
   $$\{(x_i, \mathsf{open}(S, i)) \ : \ i \in \{c, \pi(c)\}\}$$
   - 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
   $$\tilde{x}_c = \mathcal{H}(c, \tilde{x}_{i_1}, \ldots, \tilde{x}_{i_d}),$$

[12] 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.

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.

7 Pebbling game

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

1. Player 1 puts initial pebbles on some subset $U \subseteq V$ of vertices.
2. Player 2 samples a "challenge vertex" $c \in V$ according to λ.
3. Player 1 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 1 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 player 1 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 $\{G_N = (V_N, E_N) || V_N | = N \in \mathbb{N}\}$ 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 on (G_N, λ_N) 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.

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

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

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 $1 - 2^{-\Theta(N/\log N)}$.

For Theorem 1 we use the construction of Paul, Tarjan and Celoni [PTC77] which relies on superconcentrator graphs. The proof of Theorem 2 will be based on superconcentrators, random bipartite expander graphs and on the graphs of Erdös, Graham and Szemerédi [EGS75].

Remark 1 As shown in [HPV77], 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 [LT82]. Results in [LT82], however, do not immediately imply a similar bound for our interactive game.

8 Security of the PoS

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

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

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

Instantiated with the graphs from Theorem 2 it is a

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

Efficiency, measured as outlined in Section 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*

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

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

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

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

Proof. 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. □

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

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

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 Section 5.2, 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 Section 6 if instantiated with the graphs from Theorem 1 and 2, respectively.

Efficiency of the PoS eq.(7). We will now analyze the efficiency of our PoS, measuring time and storage complexity as outlined in Section 3.2. We start with 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 $\varepsilon = \Theta(1)$.

9 Proof of Theorem 1

Paul, Tarjan and Celoni [PTC77] 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 $|U| \leq \frac{1}{2}cm_i$ then $|X_U| \geq \frac{1}{2}cm_i$.*

Proof. Assume that $|U| \leq \frac{1}{2}cm_i$ 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 from 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 $|U| + \frac{1}{2}cm_i \leq cm_i$ pebbles at each time instance. By Lemma 1, this is a contradiction. □

We can now prove Theorem 1. Consider $N \geq n_1$, and let i be the largest integer such that $n_i \leq N$. 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

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

10 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;
- $\mathit{\Pi}_G(X)$ 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.

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 $\mathit{\Pi}^+_G(X) = \mathit{\Pi}_G(X) \cap V^+$ to be the set of input vertices from which set X is reachable in G. If $X = \{v\}$ then we write the sets as $\mathit{\Pi}_G(v)$ and $\mathit{\Pi}^+_G(v)$.

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. $C_{(m)}$ in Definition 2 below).

10.1 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 [Vad12]; we refer to Section 10.3 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 $A \subset V^+$, $B \subseteq V^-$ 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 [Val76]; they used $(270 + o(1))m$ edges. The constant was successively improved in a long series of works. To our knowledge, the current best known construction is due to Schöning [Sch06]; it uses $(28 + o(1))m$ edges, and relies on a probabilistic argument. There are also explicit constructions of linear superconcentrators, e.g. by Alon and Capalbo [AC03] with $(44 + o(1))m$ edges.

Our third tool is graphs of Erdös, Graham and Szemerédi [EGS75] with *dense long paths.*

Theorem 4 ([EGS75]). *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 [EGS75] 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[13].

We can also show the following.

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 $[d^*/2, d^*]$. 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

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

[13] More precisely, the construction in [EGS75] 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.

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 $|X'| \leq \eta t$ 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, $|X| \leq \eta t$. 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.

□

10.2 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 $G^d_{(m,t)} \in \mathcal{G}(2mt, d \log t)$ (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 [DNW05]. 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 [PTC77]). 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 τ' is sufficiently small then there can be at most ηt layers with more than τm initial pebbles (for some 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.

10.3 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 [Vad12] 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 10.5. [14]

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 $|U_j| \leq |V_j \cap U| + |B_j| \leq \frac{1}{2}\alpha m + \frac{1}{2}\alpha m = \alpha m$.

[14] Note that the robust expansion property has been formulated in [DNW05, 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 [AC88,Upf92]. 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:

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

□

10.4 Proof of Theorem 2: a wrap-up

Using Theorem 6 and a result from [LT82], 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 $\delta > 0$.

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 2.3.1 in [LT82].

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.

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 $\epsilon > 0$.

10.5 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 [AG89]).

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*

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

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). [15]

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.

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.

[15] 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 [Bas81].

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*

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

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

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

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 $G \backslash (A \cup B)$ 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 $|X| \leq k_{\min} - 1$ 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 $\hat{G} \backslash \hat{B}$ is a $(k_{\min} - 1, c)$-expander. To prove the lemma, it thus suffices to show that $|\hat{B}| \leq k_{\min} - 1$. 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 $k_{\min} \leq |\hat{B}'| \leq 2k_{\min} \leq k_{\max}$. We have $|\hat{B}'| \in K$, so the $(K, \gamma d)$-expansion property of $\hat{G}$ implies that $|\hat{\pi}(\hat{B}')| \geq \gamma d |\hat{B}'|$. However, by inspecting the algorithm above we conclude that $|\hat{\pi}(\hat{B}')| < \gamma d |\hat{B}'|$ - a contradiction.

Proof of Lemma 4 Denote $\alpha' = \lfloor \alpha m \rfloor / m \leq \alpha$. 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 $|\pi(v) \cap A| \geq (\delta - \gamma)d$. Each node $u \in \pi(v)$ falls in A with probability $\alpha' = |A|/m$, therefore

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

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

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

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 [Vad12], 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,

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

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

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

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.

10.6 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 ([PTC77]). *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. □

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 $G \setminus (A - B)$ 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.

Acknowledgements

We'd like to thank Moni Naor for pointing out the problem with the simple solution discussed in Section A.

References

[ABC+07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Song, *Provable data possession at untrusted stores*, ACM CCS 07(Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, eds.), ACM Press, October 2007, pp. 598–609.

[ABW03] Martín Abadi, Michael Burrows, and Ted Wobber, *Moderately hard and memory-bound functions*, NDSS 2003, The Internet Society, February 2003.

[AC88] N. Alon and F. Chung, *Explicit construction of linear sized tolerant networks*, Discrete Math. **72** (1988), 15–19.

[AC03] N. Alon and M. Capalbo, *Smaller explicit superconcentrators*, Internet Mathematics **1(2)** (2003), 151–163.

[AG89] R. Arratia and L. Gordon, *Tutorial on large deviations for the binomial distribution*, Bulletin of Mathematical Biology **51(1)** (1989), 125–131.

[And13] Nate Anderson, *Mining Bitcoins takes power, but is it an "environmental disaster"?*, April 2013, `http://tinyurl.com/cdh95at`.

[Bac97] Adam Back, *Hashcash. popular proof-of-work system.*, 1997, `http://bitcoin.org/bitcoin.pdf`.

[Bas81] L. A. Bassalygo, *Asymptotically optimal switching circuits*, Problems of Information Transmission **17** (1981), no. 3, 206–211.

[BG08] Boaz Barak and Oded Goldreich, *Universal arguments and their applications*, SIAM J. Comput. **38** (2008), no. 5, 1661–1694.

[BJO09] Kevin D. Bowers, Ari Juels, and Alina Oprea, *Proofs of retrievability: theory and implementation*, CCSW, 2009, pp. 43–54.

[BR93] Mihir Bellare and Phillip Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, ACM CCS 93(V. Ashby, ed.), ACM Press, November 1993, pp. 62–73.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi, *The random oracle methodology, revisited (preliminary version)*, 30th ACM STOC, ACM Press, May 1998, pp. 209–218.

[DGN03] Cynthia Dwork, Andrew Goldberg, and Moni Naor, *On memory-bound functions for fighting spam*, CRYPTO 2003 (Dan Boneh, ed.), LNCS, vol. 2729, Springer, August 2003, pp. 426–444.

[DN93] Cynthia Dwork and Moni Naor, *Pricing via processing or combatting junk mail*, CRYPTO'92 (Ernest F. Brickell, ed.), LNCS, vol. 740, Springer, August 1993, pp. 139–147.

[DNW05] Cynthia Dwork, Moni Naor, and Hoeteck Wee, *Pebbling and proofs of work*, CRYPTO 2005 (Victor Shoup, ed.), LNCS, vol. 3621, Springer, August 2005, pp. 37–54.

[Dou02] John R. Douceur, *The sybil attack*, IPTPS, 2002, pp. 251–260.

[DPML+03] 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*, Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on, 2003, pp. 397–406.

[EGS75] Paul Erdös, Ronald L. Graham, and Endre Szemerédi, *On sparse graphs with dense long paths*, Tech. Report STAN-CS-75-504, Stanford University, Computer Science Dept., 1975.

[FFM] Matthew Franklin, , Matthew K. Franklin, and Dahlia Malkhi, *Auditable metering with lightweight security*, Journal of Computer Security, Springer-Verlag, pp. 151–160.

[Gay13] Chris Gayomali, *Want to make money off Bitcoin mining? Hint: Don't mine*, April 2013, `http://tinyurl.com/bv43x77`.

[GJM02] Philippe Golle, Stanislaw Jarecki, and Ilya Mironov, *Cryptographic primitives enforcing communication and storage complexity*, FC 2002(Matt Blaze, ed.), LNCS, vol. 2357, Springer, March 2002, pp. 120–135.

[GK03] Shafi Goldwasser and Yael Tauman Kalai, *On the (in)security of the Fiat-Shamir paradigm*, 44th FOCS, IEEE Computer Society Press, October 2003, pp. 102–115.

[Hel80] Martin E. Hellman, *A cryptanalytic time-memory trade-off*, IEEE Transactions on Information Theory **26** (1980), no. 4, 401–406.

[HPV77] John Hopcroft, Wolfgang Paul, and Leslie Valiant, *On time versus space*, Journal of the ACM **24(2)** (1977), 332–337.

[JB99] Ari Juels and John G. Brainard, *Client puzzles: A cryptographic countermeasure against connection depletion attacks*, NDSS'99, The Internet Society, February 1999.

[JJ99] Markus Jakobsson and Ari Juels, *Proofs of work and bread pudding protocols.*, Communications and Multimedia Security (Bart Preneel, ed.), IFIP Conference Proceedings, vol. 152, Kluwer, 1999, pp. 258–272.

[JK07] Ari Juels and Burton S. Kaliski Jr., *Pors: proofs of retrievability for large files*, ACM CCS 07(Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, eds.), ACM Press, October 2007, pp. 584–597.

[LT82] Thomas Lengauer and Robert E. Tarjan, *Asymptotically tight bounds on time-space trade-offs in a pebble game*, Journal of the ACM **29(4)** (1982), 1087–1130.
[Mic00] Silvio Micali, *Computationally sound proofs*, SIAM J. Comput. **30** (2000), no. 4, 1253–1298.
[MR02] Silvio Micali and Ronald L. Rivest, *Micropayments revisited*, CT-RSA 2002 (Bart Preneel, ed.), LNCS, vol. 2271, Springer, February 2002, pp. 149–163.
[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein, *Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology*, TCC 2004(Moni Naor, ed.), LNCS, vol. 2951, Springer, February 2004, pp. 21–39.
[Nak09] Satoshi Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2009, `http://bitcoin.org/bitcoin.pdf`.
[PTC77] Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni, *Space bounds for a game on graphs*, Mathematical systems theory **10(1)** (1976–1977), 239–251.
[Rog06] Phillip Rogaway, *Formalizing human ignorance*, Progress in Cryptology - VIETCRYPT 06(Phong Q. Nguyen, ed.), LNCS, vol. 4341, Springer, September 2006, pp. 211–228.
[RS96] Ronald L. Rivest and Adi Shamir, *Payword and micromint: two simple micropayment schemes*, CryptoBytes, 1996, pp. 69–87.
[Sch06] U. Schöning, *Smaller superconcentrators of density 28*, Information processing letters **98(4)** (2006), 127–129.
[Upf92] E. Upfal, *Tolerating a linear number of faults in networks of bounded degree*, Proc. 11th PODC, 1992, pp. 83–89.
[vABHL03] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford, *CAPTCHA: Using hard AI problems for security*, EUROCRYPT 2003 (Eli Biham, ed.), LNCS, vol. 2656, Springer, May 2003, pp. 294–311.
[Vad12] S. P. Vadhan, *Pseudorandomness*, Foundations and Trends in Theoretical Computer Science **7(1-3)** (2012), 1–336.
[Val76] L. G. Valiant, *Graph-theoretic properties in computational complexity*, Journal of Computer and System Sciences **13(3)** (1976), 278–285.
[WJHF04] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten, *New client puzzle outsourcing techniques for dos resistance*, Proceedings of the 11th ACM conference on Computer and communications security (New York, NY, USA), CCS '04, ACM, 2004, pp. 246–256.

A Time-Memory Tradeoffs: Why Constructing a PoS is Non-Trivial

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 [Hel80]. 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 [Hel80] 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

[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.

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 Bitcoin

A recent exciting application of PoW is Bitcoin, a decentralized digital currency first described in October 2008 [Nak09] which has become widely popular by now. An important problem any digital currency system has to address is double spending, that is, the possibility of a cheating party spending the same digital coins in two or more transactions. Not having a central authority, Bitcoin relies on the computing power of its user-base to avoid double spending. A transaction takes something between 10 to 60 minutes to be confirmed, and a cheating party trying to double spend would have to invest more (or at least a comparable amount of) computational work in this timespan as the contributed computational power of all honest parties combined.

PoS could be an interesting alternative (or enhancement) to PoS in decentralized payment systems like Bitcoin. The current solution using PoW in Bitcoin requires the honest user-base to constantly waste a huge amount of computational power. Even if only CPU time of processors that would otherwise be idle was contributed, this still requires energy [Gay13]. For this reason Bitcoin has even been called by some an "environmental disaster" [And13].

Unlike PoW, PoS have the advantage that contributing comes basically for free (assuming parties have some free disk-space anyway), there is just some computational cost for initializing the dedicated disk space. It is not immediately clear how to adapt our single-prover interactive PoS to the non-interactive setting with multiple provers, as it would be necessary for the Bitcoin setting. We leave this as an open problem for future work.

C 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.