Proofs of Space: When Space Is of the Essence

Giuseppe Ateniese[1,2], Ilario Bonacina[1], Antonio Faonio[1], and Nicola Galesi[1]

[1] Sapienza - University of Rome, Italy
{ateniese,bonacina,faonio,galesi}@di.uniroma1.it
[2] Johns Hopkins University, USA

Abstract. Proofs of computational effort were devised to control denial of service attacks. Dwork and Naor (CRYPTO '92), for example, proposed to use such proofs to discourage spam. The idea is to couple each email message with a proof of work that demonstrates the sender performed some computational task. A proof of work can be either CPU-bound or memory-bound. In a CPU-bound proof, the prover must compute a CPU-intensive function that is easy to check by the verifier. A memory-bound proof, instead, forces the prover to access the main memory several times, effectively replacing CPU cycles with memory accesses.

In this paper we put forward a new concept dubbed *proof of space*. To compute such a proof, the prover must use a specified amount of space, i.e., we are not interested in the number of accesses to the main memory (as in memory-bound proof of work) but rather on the amount of actual memory the prover must employ to compute the proof. We give a complete and detailed algorithmic description of our model. We develop a full theoretical analysis which uses combinatorial tools from Complexity Theory (such as pebbling games) which are essential in studying space lower bounds.

Keywords: Space Complexity, Proof of Work, Pebbling Game, Random Oracle Model.

1 Introduction

Space has a special meaning in Computer Science. It refers to the number of cells of the working tape used by a Turing Machine (TM). While a TM computes a function, it will make several steps (relevant to time complexity) and use a certain number of tape cells (relevant to space complexity).

In [13], Dwork and Naor proposed to employ proof of work (PoW) to discourage spam and, in general, to hinder denial of service attacks. Before any action (such as sending an email), the prover must perform some work and generate a proof of it that can be efficiently verified. Proofs of work are currently being used to implement a publicly verifiable ledger for Bitcoin, where transactions are registered and verified by a community of users to avoid the double-spending problem [27]. The work performed by the prover can be CPU-bound, in which the work represents the number of steps made by a TM, or memory-bound, in which the work represents the times a TM access the working tape. The motivation

M. Abdalla and R. De Prisco (Eds.): SCN 2014, LNCS 8642, pp. 538–557, 2014.

behind memory-bound PoW is that, while CPU speed may differ significantly among distinct platforms, memory latencies vary much less across machines and may prove to be more equitable and egalitarian. We stress that memory-bound function complexity measures the number of memory accesses and does not take into account the actual amount of memory employed. That is, a TM may read and mark a single cell several times to reach a certain complexity level but it will still end up using only one cell.

In this work we define the notion of *Proof of Space* (PoSpace). PoSpace forces the prover to use at least a specified amount of memory. This means, for instance, that a TM must now use a predetermined number of distinct tape cells to be able to respond to a challenge. We will show that our PoSpace construction is also a memory-bound PoW under the definition provided in [12, 14], while in general a PoW cannot be a PoSpace under our definition. The state of the art memory-bound PoW was described in [14] by Dwork, Goldberg, and Naor. Their scheme requires both the prover and the verifier to store a large table T but they devised ways to mitigate this problem via either hash trees or public-key signatures.

We view PoSpace as a valid alternative to various flavors of PoWs. In PoSpace the spotlight is turned on the amount of space rather than on CPU cycles or memory accesses as in PoWs. In addition, PoSpace solves certain problems where PoW is not applicable. For instance, we believe PoSpace can be employed in forensic analysis or device attestation to confirm remotely that an embedded device has been successfully wiped. That is, a remote device could be instructed to respond to a wipe command with PoSpace as evidence that its functional memory is now overwritten (cf. [29]).

Straw Man Solutions. Memory-bound functions were first introduced by Abadi et al. [1]. In the main construction of memory-bound PoW given in [12], both the prover and the verifier share a large random table T. The prover must compute a function by making several memory accesses to uniformly random positions in T. Through a proper tuning of the parameters, it is also possible to force the prover to reserve a specific amount of memory. In another construction, the authors of [12] show that the verifier does not have to store T. The idea is to sign all pairs $(i, T[i])$ and then challenge the prover on ℓ positions of T. The prover will return the ℓ values $T[i]$ along with an aggregate signature that can be checked by the verifier to ensure the prover is holding the table T.

We first remark that it is possible to harness recent advances in proof of storage schemes, such as Provable Data Possession (PDP) [4] and (compact) Proof of Retrievability (POR) [31], to reduce the message complexity from $O(\ell)$ to essentially constant. This solution improves upon the one in [12] and, as long as the initialization phase is performed only once, would meet our *efficiency* requirements for PoSpace. However, proof of storage does not satisfy our definition of PoSpace since the running time of the verifier depends on the size of T. The only way to avoid linear dependency is to run a PDP-based scheme with spot checking [4], but then the prover either must use more space than required or will not access all the memory locations. Intuitively, the reason why a solution based on proof of storage will not work rests upon the interpretation of what *proof of* *space* really means. Proof of storage applied to our context satisfies the notion that *"the prover can* **access** *space"*. PoSpace instead captures the stronger notion that *"the prover can* **handle** *space"*, i.e., the prover possesses, controls, and manipulates space directly. In particular, we distinguish between a prover that can only read memory and a prover that can read and write memory. This is important because, among other things, write operations cannot be parallelized within classical computer architectures. We will provide a formal definition later and make this intuition rigorous.

We also remark that the adversarial model considered in [1, 12] contemplates the existence of a small but fast cache memory that must be saturated to force the prover to dispense with the cache and use traditional RAM memory. Thus, the constructions in [1, 12, 14] do provide a form of proof of space where the space coincides with the cache memory. But, as for proof-of-storage schemes, these schemes satisfy the weaker notion of PoSpace where the prover can only read memory.

Other Related Work. A proof of work is also known as a *cryptographic puzzle* in the computer security literature. Puzzles were devised to improve on the proposal by Back [6] and employed to thwart denial of service attacks. In particular, it is important to make them hard to precompute (see [22] and references therein). Waters et al. [34] suggest to outsource the creation of puzzles to an external secure entity. Abliz and Tznati [2] introduce the concept of network-bound puzzles where clients collect tokens from remote servers before querying the service provider. They argue that network latency provides a good solution to the resource disparity problem.

All solutions above deal with proof of effort and cannot be adapted to prove possession of space in the way it is meant and defined in this paper.

Litecoin (`litecoin.org`) is a variant of Bitcoin that employs *scrypt* [28] to certify a public ledger. *scrypt* is defined as a sequential memory-hard function and originally designed as a key derivation function, but it is used as a proof of effort in litecoin to hinder the use of specialized hardware. Technically, *scrypt* is not a memory-bound function as defined in [1] since no lower bound on the memory used by the function can be guaranteed. Thus, it is not even a PoSpace. We also note it requires both the prover and the verifier to dedicate a possibly large amount of memory, while ideally only the prover should reserve and use actual memory (as in our construction to be presented later).

Dziembowski et al. [18] have independently suggested a notion of proof of space. Their original construction generalizes the hash-based PoW of Cash [6] and does not employ the pebbling framework of [14] (cf. Appendix A of [15]). A major overhaul version of their paper later appeared on the IACR Crypto Eprint repository [15], along with ours [3]. Their new version does use pebbling and adopts techniques similar to ours.

However, there are two main differences between our work and [15] that make their work quite compelling:

1. The definition of proof of space in [15] is stronger in that it allows a two-stage protocol. The first stage can be executed once while the second stage can potentially be executed many times after the first. There are several applications that would benefit from this two-stage notion (see for example the Gmail scenario described in [15]). At the same time, however, this stronger notion is achieved in a more idealized model than the Random Oracle Model. To emphasize the differences between these two notions, we will often refer to ours as a one-stage PoSpace.
2. In [15] the authors provide two main constructions. The one that can be compared with our work achieves better pebbling complexity ($N/(\log(N)$ vs. $N/(\log(N) \cdot k)$). Namely, they are able to remove the extra k (a security parameter) through a clever technique.

The pebbling framework introduced in [14] has been used successfully in many other contexts (see for example [15–17, 23, 32]). Dziembowski et al. [16] built a leakage resilient key evolution scheme based on the pebbling framework. Their model allows an internal memory-bounded adversary that can control the update operation and leak bounded amounts of information. Smith and Zhang obtain a more efficient scheme [32], specifically, their update operation runs in time quasi-linear in the key length, rather than quadratic. The key-evolution scheme can be adapted to obtain a proof of space with efficient communication complexity but the space complexity of the verifier would not satisfy the efficiency constraint of PoSpace.

In a recent paper [23], Karvelas and Kiayias provide two efficient constructions for Proof of Secure Erasure (PoSE) as introduced by Perito and Tsudik [29]. Informally, in PoSE the prover must convince a verifier that a certain amount of memory has been *erased*. Both schemes in [23] are ingenious and one of them uses the pebbling framework. PoSE and PoSpace are closely related notions but have different requirements as stated in [23]. In addition, in PoSpace the prover must show that he can access (read/write) memory while in PoSE, intuitively, there should be the extra and necessary requirement that the memory contents before and after the protocol execution are uncorrelated.

General Ideas behind Our Protocol. We cast PoSpace in the context of delegation of computation, where a *delegator* outsources to a *worker* the computation of a function on a certain input. Securely delegating computation is a very active area [10, 11, 19, 21] thanks also to the popularity of cloud computing where weak devices use the cloud to compute heavy functions. In the case of PoSpace, a function f is first selected satisfying the property that there exists a space lower bound for any TM computing it. Then, the verifier chooses a random input x and delegates to the prover the computation of $f(x)$.

Specifically, we will turn to the class of functions derived from the *"graph labeling problem"* already used in cryptography (see for example [14, 16, 17]). Important tools in delegation of computation are interactive proof (and argument) systems. The delegation problem, as described in [26], can be solved as follows: The worker computes $y := f(x)$ and proves using an interactive proof system that indeed $y = f(x)$. While interactive proofs with statistical soundness and non-trivial savings in verification time are unlikely to exist [8, 9, 20], Kilian showed [24] that proof systems for **NP** languages with only computational soundness (i.e., argument systems [7]) and *succinctness* [1] do exist. The construction in [24] relies on Merkle Trees and PCP proofs. However, in the context of PoSpace, the PCP machinery is an overkill. In fact, the prover does not have to prove the statement $f(x) = y$, but only that a space-consuming computation was carried out. Therefore, we replace the PCP verification with an ad-hoc scheme that results in a very efficient overall construction (while PCP-based constructions are notoriously impractical).

Our Contributions. We introduce a formal definition of Proof of Space (PoSpace Definition 2) capturing the intuitive idea of proving to be able to handle (read / write) at least a specified amount of space. We provide two PoSpace protocols in the ROM. In addition, we provide a weaker form of PoSpace (wPoSpace) and prove that it is indeed a separate notion. Most of previous work on proof of storage [4, 5, 31] and on memory-bound PoW, as defined in [1, 12, 14], can somehow be adapted to meet this weaker definition but we will not elaborate on this any further in this paper.

Structure of the Paper. Section 2 contains some preliminary definitions. Section 3 contains the formal definition of PoSpace and provide two PoSpace protocols. Section 5 contains the definition of a *weak* variant of PoSpace that captures read-only provers and a separation result between PoSpace and this weaker variant.

Open Problems. It is not clear whether in general PoSpace implies the standard definition of PoW in the sense of [13] (i.e., whether PoSpace is also a PoW). The main obstacle to proving a positive result consists in showing *non-amortizability.* Roughly speaking, non-amortizability means that the "price" of computing the function for l different inputs is comparable to l times the "price" of computing the same function once. In the context of PoWs, the "price" is measured in terms of computational time. A PoSpace prover for space S requires a computational time proportional to S, thus we would like to reduce an adversary for l different protocol executions to an adversary for a single execution which spends less than S computational time. The point is that in order to carry out l executions, the prover needs S space and we need to ensure that he spends $l \times S$ computational time. But space is reusable and it may well happen that something already computed for one instance is reused to compute the proof for another instance. Nevertheless, our second construction does satisfy the definition of PoW. This is simply because we resort to the Random Oracle to ensure that two instances of the protocol are uncorrelated.

[1] I.e., the total amount of communication and the verification time are both less than their respective values required to transmit and check the NP-witness.

2 Notations and Preliminary Definitions

Graph Notation. Given a directed acyclic graph (DAG) G, the set of successors and predecessors of v in G are respectively $\Gamma^+(v)$ and $\Gamma^-(v)$. We will implicitly assume a topological ordering on its vertex set and a boolean encoding of the vertices respecting that ordering. If $\Gamma^+(v) = \emptyset$ then v is an *output-node* and $\mathbf{T}(G)$ is the set of all output-nodes of G. Analogously if $\Gamma^-(v) = \emptyset$ then v is an *input-node* and $\mathbf{S}(G)$ is the set of all input-nodes of G.

Sampling, Interactive Execution, and Space. Let $\mathcal{A}$ be a probabilistic TM. We write $y \leftarrow \mathcal{A}(x)$ to denote y sampled from the output of $\mathcal{A}$ on input x. Moreover, given $\sigma \in \{0,1\}^*$, we write $y := \mathcal{A}(x; \sigma)$ to denote the output of $\mathcal{A}$ on input x fixing the random coins to be σ. We write PPT for the class of probabilistic polynomial time algorithms. Let $\mathcal{A}$, $\mathcal{B}$ be two probabilistic interactive TMs. An interactive joint execution between $\mathcal{A}$, $\mathcal{B}$ on common input x is specified via the next message function notation: let $b_1 \leftarrow \mathcal{B}(x)$, $a_i \leftarrow \mathcal{A}(x, b_1, \ldots, b_i)$, and $b_{i+1} \leftarrow \mathcal{B}(x, a_1, \ldots, a_i)$, then $\langle \mathcal{A}, \mathcal{B} \rangle (x)$ denotes a joint execution of $\mathcal{A}$ and $\mathcal{B}$ on common input x. If the sequence of messages exchanged is $(b_1, a_1, \ldots, b_k, a_k, b_{k+1})$ we say that k is the number of *rounds* of that joint execution and we denote with $\langle \mathcal{A}, \mathcal{B} \rangle (x) = b_{k+1}$ the last output of $\mathcal{B}$ in that joint execution of $\mathcal{A}$ and $\mathcal{B}$ on common input x.

With Space$(\mathcal{A}, x)$, we denote the maximal amount of space used by the deterministic TM $\mathcal{A}$ on input x without taking into account the length of the input and output. More formally, we say that $\mathcal{A}(x)$ has space s (or Space$(\mathcal{A}, x) = s$) if and only if at most s locations on $\mathcal{A}$'s work tapes (excluding the input and the output tapes) are ever written by $\mathcal{A}$'s head during its computation on x. In the case of probabilistic TM $\mathcal{A}$, the function Space$(\mathcal{A}, x)$ is a random variable that depends on $\mathcal{A}$'s randomness.

Similarly, given two interactive TMs $\mathcal{A}$ and $\mathcal{B}$, we can define the space occupied by $\mathcal{A}$ during a joint execution on common input x as follows. Let $(b_1, a_1, \ldots, b_k, a_k, b_{k+1})$ the sequence of messages exchanged during $\langle \mathcal{A}, \mathcal{B} \rangle (x)$, then:

$$\mathrm{Space}_{\mathcal{A}}(\langle \mathcal{A}, \mathcal{B} \rangle, x) := \max_{i=1,\ldots,n} \{\mathrm{Space}\,(\mathcal{A}, x, b_1, \ldots, b_i))\}$$

As before, if $\mathcal{A}$ and $\mathcal{B}$ are probabilistic interactive TM then the function Space$_{\mathcal{A}}(\langle \mathcal{A}, \mathcal{B} \rangle, x)$ is a random variable that depends on the randomness of both $\mathcal{A}$ and $\mathcal{B}$. For simplicity, when the inputs of $\mathcal{A}$ (or $\langle \mathcal{A}, \mathcal{B} \rangle$) are clear from the context, we write Space$(\mathcal{A})$ (or Space$_{\mathcal{A}}$ $(\langle \mathcal{A}, \mathcal{B} \rangle)$).

Merkle Trees. Merkle Trees (MT) are classical tools in cryptography. A MT enables a party to succinctly commit itself to a string $\boldsymbol{l} = (l_1, \ldots, l_n)$ with $l_i \in \{0,1\}^k$.

The term "succinctly" here means that the MT-commitment has size k which is independent with respect to the size of $\boldsymbol{l}$. In a later stage, when a party opens the MT-commitment, it is guaranteed that the "opening" can yield only the string $\boldsymbol{l}$ committed before (the binding property). Moreover $\boldsymbol{l}$ can be succinctly opened location-by-location: the party can open l_i for any i giving the certificate attesting the value of l_i. Here we use the term succinctly to mean that the "*opening*", i.e., the certificate, has size $\log n \times k$ (sub-linear in n) with respect to the size of $\boldsymbol{l}$.

Abstractly, we define a Merkle Tree as a tuple of three algorithms (Gen_{CRH}, MT, Open) where the first algorithm is a key generation algorithm for a collision resistance hash (CRH) function. Suppose that Gen_{CRH} outputs a key s. Then MT takes as input s and a sequence of strings $\boldsymbol{l}$ and outputs the *commitment* C for $\boldsymbol{l}$, i.e., $C = \mathsf{MT}_s(\boldsymbol{l})$. The algorithm Open takes as input the key s, a sequence of strings $\boldsymbol{l}$, and an index i (it is denoted as $\mathrm{Open}_s(\boldsymbol{l}, i)$), and outputs the string l_i in $\boldsymbol{l}$.

Usually, the term "commitment" refers to a scheme that is both *hiding* (i.e., the receiver cannot infer any knowledge on $\boldsymbol{l}$ from the commitment) and *binding*. The MT scheme that we use does not provide the *hiding* property but we still refer to it as a commitment.

A full description of the Merkle Tree is deferred to Appendix B.

Pebbling Games with Wildcards. The following definition of *pebbling game with wildcards* is a modification of the standard definition of pebbling game that can be found, for instance, in [25]. Given a DAG $G = (V, E)$, we say that a sequence $\mathcal{P} = (P_0, \ldots, P_T)$ of subsets of V is a *pebbling sequence on G with m wildcards* if and only if $P_0 = \emptyset$ and there exists a set $W \subseteq V$ of size m such that, for each $i \in \{1, \ldots, T\}$, exactly one of the following holds:

- $P_i = P_{i-1} \cup \{v\}$ if $\Gamma^-(v) \subseteq P_{i-1} \cup W$ (*pebbling*) or
- $P_i \subseteq P_{i-1}$ (*unpebbling*).

If a set of vertexes Γ is such that $\Gamma \subseteq \bigcup_{i=0}^{T} P_i$, we say that $\mathcal{P}$ *pebbles* Γ. If $\mathcal{P}$ pebbles $\mathbf{T}(G)$ then we say that $\mathcal{P}$ is a *pebbling game on G with m wildcards*. Moreover we say that the *pebbling time* of $\mathcal{P}$ is T and the *pebbling space* of $\mathcal{P}$ is $\max_i |P_i|$. Intuitively, a pebbled node is a node for which we have made some computations. Instead, W represents *complementary* nodes, for which we have made no computations.

One of the main ingredients for the correctness of our constructions is the Pebbling Theorem [25] that proves that stacks of *superconcentrators* graphs (Pippenger [30]) have an exponential pebbling space-time trade off.

Theorem 1 (Pebbling Theorem). *There exists a family of efficiently sampleable directed acyclic graphs $\{G_{N,k}\}_{k,N\in\mathbb{N}}$ with constant fan-in d, N input nodes, N output nodes and $kN \log N$ nodes in total, such that:*

1. *Any pebbling sequence that pebbles Δ output nodes has pebbling space at most S pebbles and m wildcards, where $|\Delta| \geqslant 4S + 2m + 1$, and pebbling time T such that*
$$T \geqslant |\Delta| \left(\frac{N - 2S - m}{2S + m + 1} \right)^k .$$

2. *There exists a pebbling sequence that pebbles the graph $G_{N,k}$ which has pebbling space $N+2$ and needs time $O(kN\log N)$.*
3. *For any node $v \in V(G_{N,k})$, the incoming nodes of v and the position of v in first lexicographic topologically order of $G_{N,k}$ are computable in $O(k\log N)$.*

More details about the construction and the proof of this theorem are given in Appendix A.

Graph Labeling Problem with Faults. We adopt the paradigm where the action of pebbling a node in a DAG G is made equivalent to the action of having calculated some labeling on it. This paradigm was introduced in [14] and also recently used in [16, 17]. We make use of a Random Oracle (RO) $\mathcal{H}$ to build a labeling on G according to the pebbling rules.

Definition 1 ($\mathcal{H}$-labeling with faults). *Given a DAG G with a fixed ordering of the nodes and a Random Oracle function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$, we say that $\ell : V(G) \to \{0,1\}^k$ is a* (partial) $\mathcal{H}$-labeling of G with m faults *if and only if there exists a set $M \subseteq V(G)$ of size m such that for each $v \in V(G) \setminus M$*

$$\ell(v) := \mathcal{H}(v\|\ell(v_1)\|\ldots\|\ell(v_d)) \textit{ where } \{v_1,\ldots,v_d\} = \Gamma^-(v). \tag{1}$$

Given a label ℓ and a node v, we say that ℓ well-label *v if only if the equation (1) holds for ℓ and v.*

Our framework generalize the paradigm of [14] by introducing the concept of "faults". As it is shown in Section 3.1, dealing with "faults" is necessary because an adversary challenged on a labeling function could cheat by providing an inconsistent label on some nodes (which, indeed, are then referred to as "faults").

The use of a Random Oracle $\mathcal{H}$ provides two important benefits to our construction: First, the incompressibility of any output given the input and the evaluation of the function in many different points. Specifically, for any x and $x_1,\ldots,x_m$, the value of $\mathcal{H}(x)$ is uniformly random and independent of $\mathcal{H}(x_1),\ldots,\mathcal{H}(x_m)$. Therefore, to store $\mathcal{H}(x)$, an adversary needs space equal to the minimum between the shortest description of the input and the length of the output. In particular, notice that we do not require that the *entire* function be incompressible (this holds for a Random Oracle but it is trivially false for any *real-world* instantiation of it). Second, in order to $\mathcal{H}$-label the graph, any TM must follow a pebble strategy. In particular, to label a node v, a TM must necessarily calculate and store the label values of all the predecessors $\ell(v_1),\ldots,\ell(v_d)$ of the node v. If the graph G needs at least S pebbles to be pebbled efficiently in a pebbling game, then a TM needs to store at least S labels (i.e., RO outputs) to compute an $\mathcal{H}$-labeling of G. This general strategy is proven sound in [14] and referred to as the *Labeling Lemma.* In our context, however, we provide m degrees of freedom and, given a partial $\mathcal{H}$-labeling ℓ of G with m faults and a $\mathcal{H}$-labeling ℓ' of G, it will likely be the case that $\ell(v) \neq \ell'(v)$ for each node v that is a descendant of a not well-labeled node. For this reason, we must state a more general version of the Labeling Lemma in [14].

Lemma 1 (Labeling with Faults Lemma). *Consider a DAG G with degree d, a TM $\mathcal{A}$ with advice h, and a Random Oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ that computes an $\mathcal{H}$-labeling ℓ with m faults of G. If h is independent of $\mathcal{H}$, with overwhelming probability, there exists a pebbling sequence $\mathcal{P} = (P_1, \ldots, P_T)$ for the DAG G with m wildcards having pebbling space S such that:*

- $S \leqslant \frac{1}{k}\,\mathrm{Space}(\mathcal{A}) + d$,
- $T \leqslant (d+2)\sigma$, *where σ is the number of queries of $\mathcal{A}$ to $\mathcal{H}$.*

In particular T is a lower bound for the execution time of $\mathcal{A}$.

The proof of the Lemma above is provided in the full version of this paper [3].

3 Proof-of-Space Protocols

In this section we define the notion of PoSpace, then we provide two constructions that meet the definition. We later define a second notion of a *weak* form of PoSpace and show a separation result between the two notions. In our definition below, we allow the adversary to access extra information to model the case in which the adversary may outsource storage and computation to an external provider.

We model the write permission on the storage by providing a precomputation phase to the adversary. That is, the adversary can use as much space as needed to produce an hint. The hint, that may depend on the public parameters of PoSpace, can be read during the interactive phase (i.e., when the protocol is started). In contrast, wPoSpace does not provide the adversary with a precomputation phase.

Definition 2 (PoSpace)**.** *Consider $\Sigma = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$, where* Gen *is a* PPT *TM, and* P*,*V *are interactive* PPT *TMs. Let $k \in \mathbb{N}$ be a security parameter. Suppose that the following points hold :*

(Completeness) *For all $\mathsf{pk} \in \mathsf{Gen}(1^k)$ and for all $S \in \mathbb{N}$, $S > k$ it holds $\langle \mathsf{P}, \mathsf{V} \rangle (\mathsf{pk}, 1^S) = 1$, time complexity of* P *is $O(\mathsf{poly}(k, S))$;*

(Succinctness) *For all $S \in \mathbb{N}$, $S > k$ the time and space complexity of* V *and the message complexity of $\langle \mathsf{P}, \mathsf{V} \rangle$, as functions of k and S, are $O(\mathsf{poly}(k) \cdot \mathsf{poly}\log S)$;*

(Soundness) *For any* PPT *adversary $\mathcal{A}$ and for any* PPT *TM with advice $\mathcal{A}'$ such that $\mathsf{pk} \leftarrow \mathsf{Gen}(1^k)$ and $h \leftarrow \mathcal{A}'(\mathsf{pk}, 1^S)$, the following event:*

$$\mathbf{Snd}^{\mathcal{A},\mathcal{A}'}_{\Sigma,S}(k) := \langle \mathcal{A}(h), \mathsf{V} \rangle (\mathsf{pk}, 1^S) = 1 \ \wedge \ \mathrm{Space}_{\mathcal{A}}(\langle \mathcal{A}(h), \mathsf{V} \rangle, \mathsf{pk}, 1^S) < S$$

has negligible probability (as a function of k) for all $S \in \mathbb{N}$, $S > k$.

Then, we say that $\Sigma = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ is a (one-stage) Proof of Space (PoSpace)*. To be concise, we could say informally that $\mathcal{A}$* wins *when the event $\exists S \in \mathbb{N} : \mathbf{Snd}^{\mathcal{A},\mathcal{A}'}_{\Sigma,S}(k)$ occurs.*

Notice that in the completeness part we set just a very mild upper bound on the space complexity of P. This is done on purpose to allow comparison among different $\mathsf{PoSpace}$ protocols. In particular a useful measure on a $\mathsf{PoSpace}$ protocol $(\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ is the following *space gap*: the ratio $\mathrm{Space}(\mathsf{P}(\mathsf{pk}, 1^S))/S$.

Notice that $\mathcal{A}'$ is a space-unbounded PPT TM that models the fact that there might be information that can be efficiently computed that the space-bounded adversary $\mathcal{A}$ can exploit somehow to compromise $\mathsf{PoSpace}$.

It is easy to see that the $\mathsf{PoSpace}$ definition implicitly provides *sequential composability.* In fact, the adversary $\mathcal{A}'$ gives to $\mathcal{A}$ a hint which is a function of the public key, therefore the adversary $\mathcal{A}'$ can compute all the previous executions of the protocol "in his head".

We provide next a 4-messages $\mathsf{PoSpace}$ protocol in the Random Oracle Model (ROM) without any computational assumption. By applying the Fiat-Shamir paradigm to the scheme, we obtain a 2-message non-interactive $\mathsf{PoSpace}$.

3.1 A 4-Message PoSpace Protocol

The protocol $\Sigma_4 = (\mathsf{Gen}, \mathsf{V}, \mathsf{P})$ is described in Figure 1 and it is a 4-message protocol.

The protocol follows in some way Kilian's construction of argument systems [24]. For any string $\alpha \in \{0,1\}^*$, let $\mathcal{H}_\alpha(\cdot)$ be defined as $\mathcal{H}(\alpha\|\cdot)$. The verifier chooses a random α and asks the prover to build a $\mathcal{H}_\alpha$-labeling of graph $G_{N,k}$, where N depends on S. The purpose of α is to "reset" the Random Oracle. That is, any previous information about $\mathcal{H}$ is now useless with overwhelming probability. The labeling provides evidence that the prover has handled at least S memory cells. The prover then commits the labeling and sends the commitment to the verifier. At this point, the verifier asks the prover to open several random locations in the commitment and then it checks locally the integrity of the labeling. For a commitment C and for any node that the verifier has challenged, the prover sends what we call a *C-proof* for the node (defined next).

Definition 3 (C-proof). *Given a DAG G, a commitment C, and a Random Oracle $\mathcal{H}$, we say that a string $\boldsymbol{\pi} = (\pi_0, \ldots, \pi_d)$ is a C-*proof for a vertex $v \in V(G)$ w.r.t. $\mathcal{H}$ *if only if given $\Gamma_G^-(v) = \{w_1, \ldots, w_d\}$ the following points hold:*

1. *π_0 is a C-opening for v, let x be the value π_0 is opening to;*
2. *for each $i = 1, \ldots, d$, π_i is a C-opening for w_i, let x_i be the value π_i is opening to;*
3. *$x = \mathcal{H}(v\|x_1\|\ldots\|x_d)$.*

We omit C, G, and $\mathcal{H}$, when they are clear from the context, by saying that π is simply a proof.

In the definition of C-proof, the points 1 and 2 refer to the commitment C while point 3 ensures the integrity of the labeling. Note that the size of $\boldsymbol{\pi}$ is $O(kd \log N)$.

We remark that when TM $\mathcal{B}$ takes as input a RO $\mathcal{H}$, it is intended that $\mathcal{B}$ has oracle access to $\mathcal{H}$ and the length of the input of $\mathcal{B}$ does not take into account the (exponentially long) length of $\mathcal{H}$.

Generator Gen takes as **input** 1^k and outputs $\mathsf{pk} := (\{G_{N,k}\}_{N\in\mathbb{N}}, \mathcal{H}, s)$,
where $\{G_{N,k}\}_{N\in\mathbb{N}}$ is a family of graphs satisfying the Pebbling Theorem (Theorem 1), equipped with the natural lexicographic topological ordering on its vertex set,
$\mathcal{H}$ is a RO and s is a key for CRH H.
Common input: k, S, pk
$N := \lceil 4\gamma(d + S/k) + \gamma\rceil$, where d is the degree of $G_{N,k}$ and $\gamma \in \mathbb{R}$, $\gamma > 1$.

Verifier V		**Prover** P
1. Pick $\alpha \leftarrow \{0,1\}^k$	$\xrightarrow{\quad\alpha\quad}$	Let ℓ be a $\mathcal{H}_\alpha$-labeling of $G_{N,k}$, where $\mathcal{H}_\alpha(\cdot) := \mathcal{H}(\alpha\|\cdot)$.
	$\xleftarrow[\quad C\quad]{}$	**Commit** $C := \mathsf{MT}_s(\ell)$.
2. Pick $(v_1, \dots, v_l) \leftarrow V^l$ uniformly at random, where $l = \lfloor k \ln^2 k \log N\rfloor$ and $V = V(G_{N,k})$	$\xrightarrow{(v_1, \dots, v_l)}$	For any v_i, $\pi_i := \mathrm{Open}_s(\ell, v_i)$: For any $v_i^j \in \varGamma^-(v_i)$: $\pi_i^j := \mathrm{Open}_s\left(\ell, v_i^j\right)$ $\varPi_i := \left(\pi_i, \pi_i^1, \dots, \pi_i^d\right)$
	$\xleftarrow[(\varPi_1, \dots, \varPi_l)]{}$	**Send** $(\varPi_1, \dots, \varPi_l)$
3. Check for any $i \leqslant l$ if $\varPi_i$ is a C-proof for v_i wrt the $\mathcal{H}_\alpha$-labeling ℓ		

Fig. 1. The 4-message $\mathsf{PoSpace}$ protocol $\varSigma_4$

Theorem 2. *The protocol $\varSigma_4$ in Figure 1 is a (one-stage)* $\mathsf{PoSpace}$.

We start by giving the intuition behind the proof of the Theorem. Completeness is trivial. Succinctness follows easily from point (3) of the Pebbling Theorem (Theorem 1). For Soundness, we first prove that the protocol $\varSigma_4$ is a Proof-of-Knowledge (PoK) of a partial labeling with "few" faults. By the PoK property, we can extract from a winning adversary a partial labeling with few faults. Thus, with overwhelming probability, the adversary $\mathcal{A}$ computes a partial labeling. We then exploit the binding property of the Merkle-Tree and show that the adversary has computed the labeling during the round (1) of the protocol. By appropriately fixing the randomness of the protocol, we ensure that the hint h is independent of $\mathcal{H}$, thus meeting all the hypotheses of the Labeling Lemma (Lemma 1). We obtain then a pebbling strategy that has pebbling space and pebbling time roughly upper-bounded by the respective space and time complexity of the adversary $\mathcal{A}$. Since the adversary uses strictly less space than S

and $\mathcal{A}$ is PPT then, by our choice of the parameters, there is a contradiction with the point (1) of the Pebbling Theorem (Theorem 1).

Proof. We focus on the *Soundness*. We divide the proof in two parts. First, in the PoK Lemma, we prove that the Protocol Σ_4 is a PoK for a partial labeling with "few" faults, and that the partial labeling has been computed during the round (1), i.e., after the verifier V sent the message α and before the adversary sent the commitment message C. Then we combine the PoK Lemma with the Labeling Lemma and the Pebbling Theorem to reach a contradiction assuming that there exists a winning PPT adversary.

We stress that the knowledge extractor doesn't need to be efficient since we do not rely on any computational assumption.

Lemma 2 (PoK Lemma). *Consider the protocol $\Sigma_4 = (\mathsf{Gen}, \mathsf{V}, \mathsf{P})$, a PPT adversary $\mathcal{A}$, a PPT Turing Machine with advice $\mathcal{A}'$, a space parameter S, and a security parameter k. Sample a random $\mathsf{pk} \leftarrow \mathsf{Gen}(1^k)$ and let $h \leftarrow \mathcal{A}'(\mathsf{pk}, 1^S)$. If $\Pr\left[\mathbf{Snd}^{\mathcal{A},\mathcal{A}'}_{\Sigma_4,S}(k)\right]$ is noticeable (where the probability is taken over the randomness of pk, h and all the randomness used during the protocol execution between $\mathcal{A}$ and the verifier) then, for a noticeable probability over the choice of pk, there exist a first verifier message $\tilde{\alpha}$ and an adversary's randomness $\tilde{\rho}$ such that $\mathcal{A}(h)$ calculates an $\mathcal{H}_{\tilde{\alpha}}$-labeling ℓ of $G_{N,k}$ with m faults such that the following holds:*

- *the hint h is independent of $\mathcal{H}_{\tilde{\alpha}}$,*
- *$m = O\left(\frac{N}{\ln k}\right)$ and*
- *for any well-labeled node u in ℓ we have $(\tilde{\alpha}\|u\|\ell(u_1)\|\ldots\|\ell(u_d)) \in \mathcal{Q}$,*

where $\mathcal{Q}$ is the set of queries to $\mathcal{H}$ made by $\mathcal{A}$ when fed with randomness $\tilde{\rho}$ during round (1) of the protocol Σ_4, and $\{u_1, \ldots, u_d\} = \Gamma^-(u)$.

The proof of the Lemma above is provided in the full version of this paper [3].

By contradiction, suppose there exists an adversary $\mathcal{A}$ for the protocol Σ_4. By applying Lemma 2, we extract with noticeable probability a partial $\mathcal{H}_{\tilde{\alpha}}$-labeling, ensure that $\mathcal{A}(h)$ computed that labeling during round (1) and that the hint h is independent of $\mathcal{H}_{\tilde{\alpha}}$. Hence, satisfying the hypotheses of Lemma 1.

This gives us, with overwhelming probability, a pebbling sequence $\mathcal{P}$ of $G_{N,k}$ with m wildcards having pebbling space $S' = \frac{S}{k} + d$, $m = O(\frac{N}{\ln k})$. In addition, the pebbling time of $\mathcal{P}$ is a lower bound for the execution time of $\mathcal{A}$ during round (1). To apply Theorem 1, we fix N such that

$$N - l \geqslant 4S' + 2m + 1, \tag{2}$$

where l denotes the number of wildcards that are in $\mathbf{T}(G_{N,k})$.

Given $c \in (0, 1)$ and $\epsilon > 0$, we have that $(2+\epsilon)m + l \leqslant cN$. If we find N such that

$$N \geqslant 4S' + cN + 1,$$

(hence $N \geqslant 4\gamma S' + \gamma$, where $\gamma = 1/(1-c)$) then the inequality (2) will follow. (This is main reason why we have set $N := \lceil 4\gamma S' + \gamma \rceil$ in the Protocol Σ_4.)

By applying Theorem 1, the execution time T of $\mathcal{A}$ is such that

$$T \geqslant (N - l) \left(\frac{N - 2S' - m}{2S' + m + 1} \right)^k \geqslant c'(1 + \epsilon)^k, \tag{3}$$

where c' is a constant that is a lower bound for $N - l$. The value $(1 + \epsilon)^k$ derives from the fact that, eventually,

$$N - 2S' - m \geqslant (1 + \epsilon)(2S' + m + 1).$$

Equation (3) shows that the execution time of $\mathcal{A}$ is exponential in k. This is not possible as, by hypothesis, $\mathcal{A}$ is PPT.

On the Space Gap of the Protocol. The prover algorithm can be implemented basically in two ways. The most natural implementation is to first build the labeling for all the nodes in the graph and then apply the Merkle Tree, while keeping in memory the labeling which is reused during the second phase of the prover algorithm. Through this algorithm, the space gap of the prover is $O(\log S)$ while the time complexity is essentially dominated by the one labeling phase. Another way to implement the prover algorithm is by computing the labeling and the Merkle Tree *simultaneously.* In this way the algorithm can reuse space resulting in a strategy with space gap $O(1)$. Note however that, in this implementation, the prover must build the labeling twice (once for the commitment and then during the challenge phase).

4 A 2-Messages PoSpace Protocol

We apply the standard *Fiat-Shamir* paradigm to the PoSpace scheme given in Section 3.1 by using two independent Random Oracles $\mathcal{H}, \mathcal{L}$:

- $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ is used by the prover for the labeling of the graph;
- $\mathcal{L} : \{0,1\}^k \to V^l$ given the commitment C as input, it yields the second verifier's message (of the protocol Σ_4).

Let $(\Sigma_4.\mathsf{Gen}, \Sigma_4.\mathsf{P}, \Sigma_4.\mathsf{V})$ the PoSpace defined in Figure 1. We define in Figure 2 a 2-messages PoSpace: we call $\Sigma_2 = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ this protocol. Furthermore, let $\mathcal{H}' : \{0,1\}^* \to \{0,1\}^k$ be a RO then the function $f(x) := \Sigma_2.\mathsf{P}(\mathsf{pk}, 1^S, \mathcal{H}'(x))$ is a non-interactive PoSpace that satisfies the syntactic definition of PoWs in [13].

Theorem 3. *The Protocol Σ_2 in Figure 2 is a* PoSpace.

The proof of the theorem 3 is provided in the full version of this paper [3].

5 *Weak* Proof of Space

The concept of proof of space can lead to multiple interpretations. The main interpretation formalized in the PoSpace definition requires that the prover can

Generator Gen on **input** 1^k returns $\mathsf{pk}' := (\mathsf{pk}, \mathcal{L})$,
where $\mathsf{pk} \leftarrow \Sigma_4.\mathsf{Gen}(1^k)$ and $\mathcal{L}$ is a RO
Common input: k, S, pk

Verifier V | **Prover** P

1. Pick $\alpha \leftarrow \{0,1\}^k$ $\xrightarrow{\alpha}$ $C := \Sigma_4.\mathsf{P}(\mathsf{pk}, 1^S, \alpha)$
$\boldsymbol{v} := \mathcal{L}(C)$

$\xleftarrow[(C, \Pi)]{}$ $\Pi := \Sigma_4.\mathsf{P}(\mathsf{pk}, 1^S, \alpha, \boldsymbol{v})$

2. Let $\boldsymbol{v} = \mathcal{L}(C)$
Check $\Sigma_4.\mathsf{V}(\mathsf{pk}, 1^S, \alpha, C, \boldsymbol{v}, \Pi) = 1$

Fig. 2. The 2-messages (one-stage) PoSpace protocol $\Sigma_2 = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$

handle (i.e., read/write) space. In this section we provide a definition for a weaker alternative of PoSpace we call *weak* Proof of Space (wPoSpace). This captures the property that the prover can just access space and formalizes what could effectively be achieved by properly adapting previous work on proof of storage [4, 5, 31] and on memory-bound PoW as defined in [1, 12](where the adversary model contemplates the existence of cache memory). The definition of wPoSpace is similar to the Definition of PoSpace (Definition 2) (the only change is in the Soundness part). We will provide a protocol which is a wPoSpace but not a PoSpace, hence wPoSpace is a strictly weaker notion than PoSpace.

Definition 4 (wPoSpace)**.** *Consider* $\Sigma = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *where* Gen *is a* PPT *TM, and* P*,* V *are interactive* PPT *TMs. Let* $k \in \mathbb{N}$ *be a security parameter and* $\mathsf{pk} \leftarrow \mathsf{Gen}(1^k)$*. Suppose that the following points hold for all* $S \in \mathbb{N}$*,* $S > k$*:*

(Completeness) and (Succinctness) *the same as in the* PoSpace *definition.*
(weak-Soundness) *For any* PPT *adversary* $\mathcal{A}$*, the following event:*

$$\mathbf{wSnd}^{\mathcal{A}}_{\Sigma,S}(k) := \langle \mathcal{A}, \mathsf{V} \rangle (\mathsf{pk}, 1^S) = 1 \wedge \mathrm{Space}_{\mathcal{A}}(\langle \mathcal{A}, \mathsf{V} \rangle, \mathsf{pk}, 1^S) < S,$$

has negligible probability (as a function of k*).*

Then we say that Σ *is a (one-stage)* Weak Proof of Space (wPoSpace)*.*

In the Soundness of PoSpace, the adversary can take advantage of an unbounded space machine which is then unavailable during the protocol execution. In the Soundness of wPoSpace, instead, this is disallowed. Notice that a wPoSpace's adversary can perform some precomputation before the execution of the protocol (i.e., before sending/receiving the first message), however, such a precomputation cannot exceed the space bound given.

Theorem 4. *There exists a protocol which is a* wPoSpace *but not a* PoSpace*.*

Proof. We start by providing a protocol which is not a $\mathsf{PoSpace}$. Consider the protocol $\varSigma_4$ in Figure 1 where the first message α sent by V is always the same, say 0^k. We call $\varSigma_3$ this modified version of $\varSigma_4$. The protocol $\varSigma_3$ is not a $\mathsf{PoSpace}$. For any k and $S \in \mathbb{N}$, consider the hint h which is the $\mathcal{H}_{0^k}$-labeling of $G_{N,k}$ plus the complete Merkle-Tree of that labeling. We define an adversary that sends the commitment C that is in h and, for any verifier's second message $\boldsymbol{v}$, reads the right answer from h. That adversary needs to access h in *read-only* mode, hence without using any additional working space.

The protocol $\varSigma_3$ is a $\mathsf{wPoSpace}$. The structure of the proof is the same as the one of Theorem 2. We provide a particular case of the PoK Lemma (Lemma 2) where the hint h is the empty string and $\tilde{\alpha}$ is 0^k. For the sake of clarity, we restate the PoK Lemma for this particular setting.

Lemma 3. *Consider the protocol $\varSigma_3 = (\mathsf{Gen}, \mathsf{V}, \mathsf{P})$, a* PPT *adversary $\mathcal{A}$, a space parameter S and a security parameter k such that $\mathsf{pk} \leftarrow \mathsf{Gen}(1^k)$.*

If $\Pr\left[\mathbf{wSnd}^{\mathcal{A}}_{\varSigma_3,S}(k)\right]$ *is noticeable, then there exists a randomness $\tilde{\rho}$ such that $\mathcal{A}$ fed with the randomness $\tilde{\rho}$ during the round (1) of the protocol $\varSigma_3$ calculates an $\mathcal{H}_{0^k}$-labeling ℓ of $G_{N,k}$ with m faults such that the following holds:*

- *$m = O\left(\frac{N}{\ln k}\right)$ and*
- *for any well-labeled node u in ℓ we have $(\tilde{\alpha}\|u\|\ell(u_1)\|\dots\|\ell(u_d)) \in \mathcal{Q}$,*

where $\mathcal{Q}$ is the set of queries to $\mathcal{H}$ made by $\mathcal{A}$ when fed with randomness $\tilde{\rho}$ until the end of round (1) and $\{u_1, \dots, u_d\} = \varGamma^-(u)$.

By contradiction, suppose there exists an adversary $\mathcal{A}$ for the protocol $\varSigma_3$. By applying Lemma 3, we extract a partial $\mathcal{H}_{0^k}$-labeling, ensuring that $\mathcal{A}$ computed that labeling during round (1) (hence satisfying the hypotheses of Lemma 1).

This gives us, with overwhelming probability, a pebbling sequence $\mathcal{P}$ of $G_{N,k}$ with m wildcards and with pebbling space $S' = \frac{S}{k} + d$, $m = O(\frac{N}{\ln k})$. The pebbling time of $\mathcal{P}$ is a lower bound for the execution time of $\mathcal{A}$ before round (2). The rest of the proof follows exactly the same structure of the proof of Theorem 2 and it is therefore omitted. □

Acknowledgements. We are grateful to Krzysztof Pietrzak for his insightful comments and suggestions.

References

1. Abadi, M., Burrows, M., Manasse, M., Wobber, T.: Moderately hard, memory-bound functions. ACM Trans. Internet Technol. 5(2), 299–327 (2005)
2. Abliz, M., Znati, T.: A guided tour puzzle for denial of service prevention. In: ACSAC, pp. 279–288. IEEE Computer Society (2009)
3. Ateniese, G., Bonacina, I., Faonio, A., Galesi, N.: Proofs of space: When space is of the essence. Cryptology ePrint Archive, Report 2013/805 (2013), `http://eprint.iacr.org/`

4. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2007, pp. 598–609. ACM, New York (2007)
5. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 319–333. Springer, Heidelberg (2009)
6. Back, A.: Hashcash - a denial of service counter-measure. Technical report (2002)
7. Barak, B., Goldreich, O.: Universal arguments and their applications. In: IEEE Conference on Computational Complexity, pp. 194–203. IEEE Computer Society (2002)
8. Boppana, R.B., Hastad, J., Zachos, S.: Does co-np have short interactive proofs? Inf. Process. Lett. 25(2), 127–132 (1987)
9. Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.): ICALP 2005. LNCS, vol. 3580. Springer, Heidelberg (2005)
10. Canetti, R., Riva, B., Rothblum, G.N.: Refereed delegation of computation. Information and Computation 226, 16–36 (2013)
11. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved Delegation of Computation Using Fully Homomorphic Encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010)
12. Dwork, C., Goldberg, A.V., Naor, M.: On memory-bound functions for fighting spam. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 426–444. Springer, Heidelberg (2003)
13. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993)
14. Dwork, C., Naor, M., Wee, H.: Pebbling and proofs of work. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 37–54. Springer, Heidelberg (2005)
15. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. Cryptology ePrint Archive, Report 2013/796 (2013), http://eprint.iacr.org/
16. Dziembowski, S., Kazana, T., Wichs, D.: Key-evolution schemes resilient to space-bounded leakage. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 335–353. Springer, Heidelberg (2011)
17. Dziembowski, S., Kazana, T., Wichs, D.: One-time computable self-erasing functions. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 125–143. Springer, Heidelberg (2011)
18. Dziembowski, S., Pietrzak, K., Faust, S.: Proofs of space and a greener bitcoin. Talk presented at Workshop on Leakage, Tampering and Viruses, Warsaw 2013 (2013)
19. Fiore, D., Gennaro, R.: Publicly verifiable delegation of large polynomials and matrix computations, with applications. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012, p. 501 (2012)
20. Goldreich, O., Hastad, J.: On the complexity of interactive proofs with bounded communication. Information Processing Letters (1998)
21. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC 2008, pp. 113–122. ACM, New York (2008)
22. Juels, A., Brainard, J.G.: Client puzzles: A cryptographic countermeasure against connection depletion attacks. In: NDSS. The Internet Society (1999)

23. Karvelas, N.P., Kiayias, A.: Efficient Proofs of Secure Erasure. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 526–543. Springer, Heidelberg (2014)
24. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, STOC 1992, pp. 723–732. ACM, New York (1992)
25. Lengauer, T., Tarjan, R.E.: Asymptotically tight bounds on time-space trade-offs in a pebble game. Journal of the ACM (JACM) 29(4), 1087–1130 (1982)
26. Micali, S.: Computationally sound proofs. SIAM Journal on Computing 30(4), 1253–1298 (2000)
27. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (May 2009)
28. Percival, C.: Stronger key derivation via sequential memory-hard functions. Presented at BSDCan 2009 (2009)
29. Perito, D., Tsudik, G.: Secure code update for embedded devices via proofs of secure erasure. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 643–662. Springer, Heidelberg (2010)
30. Pippenger, N.: Superconcentrators. SIAM J. Comput. 6(2), 298–304 (1977)
31. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)
32. Smith, A., Zhang, Y.: Near-linear time, leakage-resilient key evolution schemes from expander graphs. IACR Cryptology ePrint Archive, 2013:864 (2013)
33. Tompa, M.: Time-space tradeoffs for computing functions, using connectivity properties of their circuits. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC 1978, pp. 196–204. ACM, New York (1978)
34. Waters, B., Juels, A., Alex Halderman, J., Felten, E.W.: New client puzzle outsourcing techniques for dos resistance. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, pp. 246–256. ACM, New York (2004)

A Proof of the Pebbling Theorem (Theorem 1)

For the sake of convenience we re-write the statement of that theorem here. For the definitions of pebbling games, pebbling space and pebbling time we refer the reader to Section 2.

Pebbling Theorem (Theorem 1). *There exists a family of efficiently sampleable directed acyclic graphs $\{G_{N,k}\}_{k,N\in\mathbb{N}}$ with constant fan-in d, N input nodes, N output nodes and $kN\log N$ nodes in total, such that:*

1. *Any pebbling sequence that pebbles $\varDelta$ output nodes has pebbling space at most S pebbles and m wildcards, where $|\varDelta| \geqslant 4S + 2m + 1$, and pebbling time T such that*
$$T \geqslant |\varDelta| \left(\frac{N-2S-m}{2S+m+1}\right)^k .$$
2. *There exists a pebbling sequence that pebbles the graph $G_{N,k}$ which has pebbling space $N+2$ and needs time $O(kN\log N)$.*

3. *For any node $v \in V(G_{N,k})$, the incoming nodes of v and the position of v in first lexicographic topologically order of $G_{N,k}$ are computable in $O(k \log N)$.*

The family of graphs $G_{N,k}$ we build is made from DAGs that are k layered stack of superconcentrators [30] with N inputs and N outputs.

Definition 5 (superconcentrator). *We say that a directed acyclic graph $G = (V, E)$ is an N-*superconcentrator *if and only if*

1. $|\mathbf{S}(G)| = |\mathbf{T}(G)| = N$,
2. *for each $S \subseteq \mathbf{S}(G)$ and for each $T \subseteq \mathbf{T}(G)$ such that $|S| = |T| = \ell$ there exist ℓ vertex-disjoint paths from S to T, i.e. paths such that no vertex in V is in two of them.*

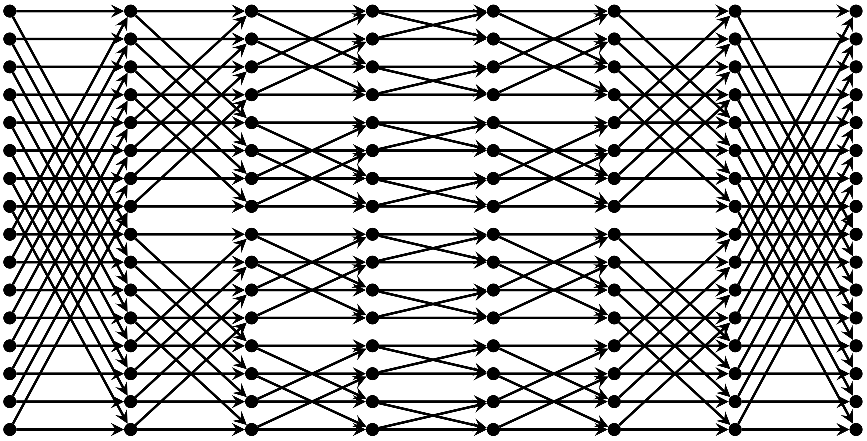

Fig. 3. An example of 16-superconcentrator: the Butterfly Graph

Definition 6 (stack of superconcentrators). *Given an N-superconcentrator G we define the graph $G^{\times k}$, the* stack of k copies of G, *inductively as follows: $G^{\times 1} = G$ and $G^{\times k}$ is obtained from $G^{\times(k-1)}$ and G by first renaming the vertexes of G so that they do not appear in the vertexes of $G^{\times(k-1)}$, obtaining a graph G', and then by identifying $\mathbf{T}(G^{\times(k-1)})$ with $\mathbf{S}(G')$.*

Not any k layered stack of superconcentrators satisfies points 2 and 3 of the Pebbling Theorem. To obtain those properties we use the following well-known superconcentrator: the *Butterfly Graph B_N* built by putting together two FFT graphs with N inputs and outputs [33]. An example of such construction (for $N = 16$) is given in Figure 3.

It is easy to see that the family of graphs $G_{N,k} := B_N^{\times k}$ satisfies points 2 and 3 of the Pebbling Theorem and it is a k layered stack of N-superconcentrators. It remains to prove only point 1.

For superconcentrator graphs we can prove a tradeoff between pebbling space and pebbling time, even for pebbling games with wildcards. This result is based on a generalization of the *"Basic Lower Bound Argument"* (BLBA) by Tompa [33].

Lemma 4 (BLBA with wildcards). *Consider an N-superconcentrator G, a set $M \subseteq \mathbf{T}(G)$, and a pebbling sequence $\mathcal{P} = (P_0, \ldots, P_\ell)$ with m wildcards that pebbles M. Let A be the set of all elements of $\mathbf{S}(G)$ pebbled and unpebbled at some point in $\mathcal{P}$. If $|M| \geqslant |P_0| + |P_\ell| + m + 1$ then $|A| \geqslant N - |P_0| - |P_\ell| - m$.*

Proof. By contradiction suppose that $|A| < N - |P_0| - |P_\ell| - m$, this means that $|A^c| \geqslant |P_0| + |P_\ell| + m + 1$ and every element in A^c, by definition, is not pebbled and unpebbled. Take any $B \subseteq A^c$ such that $|B| = |M|$ then, as G is an N-superconcentrator, we have $\pi_1, \ldots, \pi_{|M|}$ vertex disjoint paths from B to M. Let W be a set of m faults for $\mathcal{P}$. By construction $|M| > |P_0 \cup P_\ell \cup W|$ thus we have some path π from some element v of B to some element w of M such that $\pi \cap (P_0 \cup P_\ell \cup W) = \emptyset$. By definition of A^c, and hence of B, v is not pebbled and unpebbled and $v \notin P_0 \cup P_\ell \cup W$, thus $v \notin P_i$ for each $i \in [\ell]$. As $\pi \cap (P_0 \cap W) = \emptyset$, we must have that $\pi \cap P_i = \emptyset$ for each $i \in [\ell]$. But then the vertex w is not in $\bigcup_{i \in [\ell]} P_i$ contradicting the fact that $\mathcal{P}$ pebbles M. □

Theorem 5 (Pebbling Theorem (point 1)). *Let $G^{\times k}$ be a stack of k copies of an N-superconcentrator G, $\Delta \subseteq \mathbf{T}(G^{\times k})$ and $\mathcal{P}$ a pebbling sequence for $G^{\times k}$ with m wildcards that pebbles Δ. Let S be the pebbling space of $\mathcal{P}$ and T the pebbling time of $\mathcal{P}$. If $|\Delta| \geqslant 4S + 2m + 1$ then*

$$T \geqslant |\Delta| \left(\frac{N - 2S - m}{2S + m + 1} \right)^k .$$

Proof. Let $G_1, \ldots, G_k$ be the k copies of G that form $G^{\times k}$. Suppose we want to pebble a set $M \subseteq \mathbf{T}(G_i)$ such that $|M| \geqslant 2S + m + 1$ and let $A_1, \ldots, A_n$ be disjoint subsetes of M such that A_1 are the first $2S + m + 1$ elements of M to be pebbled in $\mathcal{P}$, A_2 are the second $2S + m + 1$ elements of M to be pebbled in $\mathcal{P}$ and so on. Clearly we have that the number of these sets is $n = \left\lfloor \frac{|M|}{2S+m+1} \right\rfloor$. By Lemma 4 we have that for each A_j there exists a set $\beta(A_j)$ in $\mathbf{S}(G_i)$, whose elements are pebbled and unpebbled, of size at least $N - 2S - m$. Thus we have that the total number of elements in $\mathbf{S}(G_i)$ that have been pebbled and unpebbled is at least $(N - 2S - m) \left\lfloor \frac{|M|}{2S+m+1} \right\rfloor$. Notice now that each $\beta(A_j) \in \mathbf{T}(G_{i-1})$ hence we can reapply the argument above to it provided that for each j, $|\beta(A_j)| \geqslant N - 2S - m \geqslant 2S + m + 1$ (and this is implied by the fact that $N \geqslant |\Delta| \geqslant 4S + 2m + 1$). Thus starting from $\mathbf{T}(G_k) = \mathbf{T}(G^{\times k})$ we can prove by induction, going back to G_1, that at least

$$|\Delta| \left(\frac{N - 2S - m}{2S + m + 1} \right)^k$$

actions of pebbling and unpebbling take place on $\mathbf{S}(G_1) = \mathbf{S}(G^{\times k})$. As each action is an elementary step in the pebbling game $\mathcal{P}$ we have that the result follows. □

B Merkle Tree

Consider the complete binary tree T over n leaves. Without loss of generality we can assume that n is a power of 2 by using padding if necessary. Label the i-th leaf with l_i, then, use a collision resistant hash (CRH) function $\mathsf{H}_s : \{0,1\}^{2k} \to \{0,1\}^k$ with random seed s to propagate the labeling $\boldsymbol{l}$ of the leaves to a labeling ϕ along the tree T up to the root according to the following rules:

1. $\phi(v) := l_i$ if v is the i-th leaf of T,
2. $\phi(v) := \mathsf{H}_s(\phi(v_1) \| \phi(v_2))$ where v_1, v_2 are the two children of v.

Then the *commitment* for a string $\boldsymbol{l}$ is the label $\phi(r)$ of the root of the tree: we denote it with $\mathsf{MT}_s(\boldsymbol{l})$. The commitment is succinct in fact it has size k which is independent of n.

An *opening for the i-th element in the sequence* $\boldsymbol{l}$ is an ordered sequence formed by elements of $\{0,1\}^k$ that are intended to be the label l_i of the i-th leaf of T together with the labels $\phi(v_j)$ for each v_j that is a sibling of vertices in the path from the root of the tree to the i-th leaf. An example of opening is shown in Figure 4.

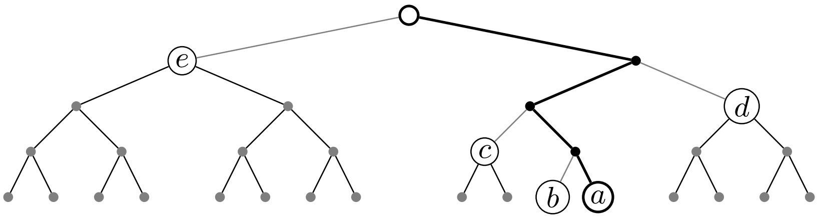

Fig. 4. An example of opening: $\boldsymbol{c} = (a, b, c, d, e)$

We denote with $\text{Open}_s(\boldsymbol{l}, i)$ the opening formed by l_i together with all the $\phi(v_j)$. Notice that from $\text{Open}_s(\boldsymbol{l}, i)$ it is possible to compute $\phi(w)$ for each vertex w in the path from the root to the leaf at the i-th position. In particular it is efficient to compute $\mathsf{MT}_s(\boldsymbol{l})$. The opening is succinct in fact it has size $k \cdot \log n$ which is (almost) independent of n. In general, given an opening $\boldsymbol{c} = (c_1, \ldots, c_{\log n})$ for some position i we can think of it as equivalent to $\text{Open}_s(\boldsymbol{l}, i)$ and compute the label of the root according to the given $\boldsymbol{c}$ and position i. If this value is equal to $\mathsf{MT}_s(\boldsymbol{l})$, we say that $\boldsymbol{c}$ *opens the i-th position to* c_1. As H is a CRH function, then for any i, it is guaranteed that $\mathsf{MT}_s(\boldsymbol{l})$ can open the i-th position only to l_i.